(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 860 267 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.08.1998 Bulletin 1998/35

(51) Int. Cl.[6]: **B29C 67/00**, G01J 1/42, H01S 3/00

(21) Application number: **96120757.8**

(22) Date of filing: **17.04.1989**

(84) Designated Contracting States:
AT BE CH DE ES FR GB GR IT LI LU NL SE

(30) Priority: **08.11.1988 US 268816**
08.11.1988 US 268837
08.11.1988 US 268907
08.11.1988 US 269801
18.04.1988 US 182830

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
89303780.4 / 0 375 097

(71) Applicant: **3D SYSTEMS, INC.**
Valencia California 91355 (US)

(72) Inventors:
- **Spence, Stuart Thomas**
  South Pasadena, CA 91030 (US)
- **Almquist, Thomas**
  San Gabriel, CA 91775 (US)
- **Tarnoff, Harry L.**
  Van Nuys, CA 91411 (US)

(74) Representative:
Geissler, Bernhard, Dr. jur., Dipl.-Phys. et al
Patent- und Rechtsanwälte
Bardehle . Pagenberg . Dost . Altenburg .
Frohwitter . Geissler & Partner
Postfach 86 06 20
81633 München (DE)

Remarks:
- This application was filed on 23-12-1996 as a divisional application to the application mentioned under INID code 62.
- A request for correction of discrepancies in the claims (6, 29 and 30) has been filed pursuant to Rule 88 EPC. A decision on the request will be taken during the proceedings before the Examining Division (Guidelines for Examination in the EPO, A-V, 3.).

(54) Stereolithographic beam profiling

(57) An apparatus and a method for profiling the intensity of a beam and thus measuring the overall intensity and power of a beam are disclosed that have particular use in stereolithography. A beam sensor comprising a pinhole in a plate and a photodetector behind the pinhole measures the intensity of portions of a beam as the beam is moved over the beam sensor. Software associated with the sensors in a computer controls the scanning mechanism for the beam so that the beam is shifted to find the pinhole and move across it in order to develop the intensity profile. The invention can be used to detect drift in the scanning mechanism, determine the focus of the beam, and predict the depth and width of photopolymer cured by the beam. A related apparatus and method for calibrating and normalizing a stereolithographic apparatus is described, and a related apparatus and method for correcting for drift in production of objects by stereolithography, is also described.

EP 0 860 267 A2

(Cont. next page)

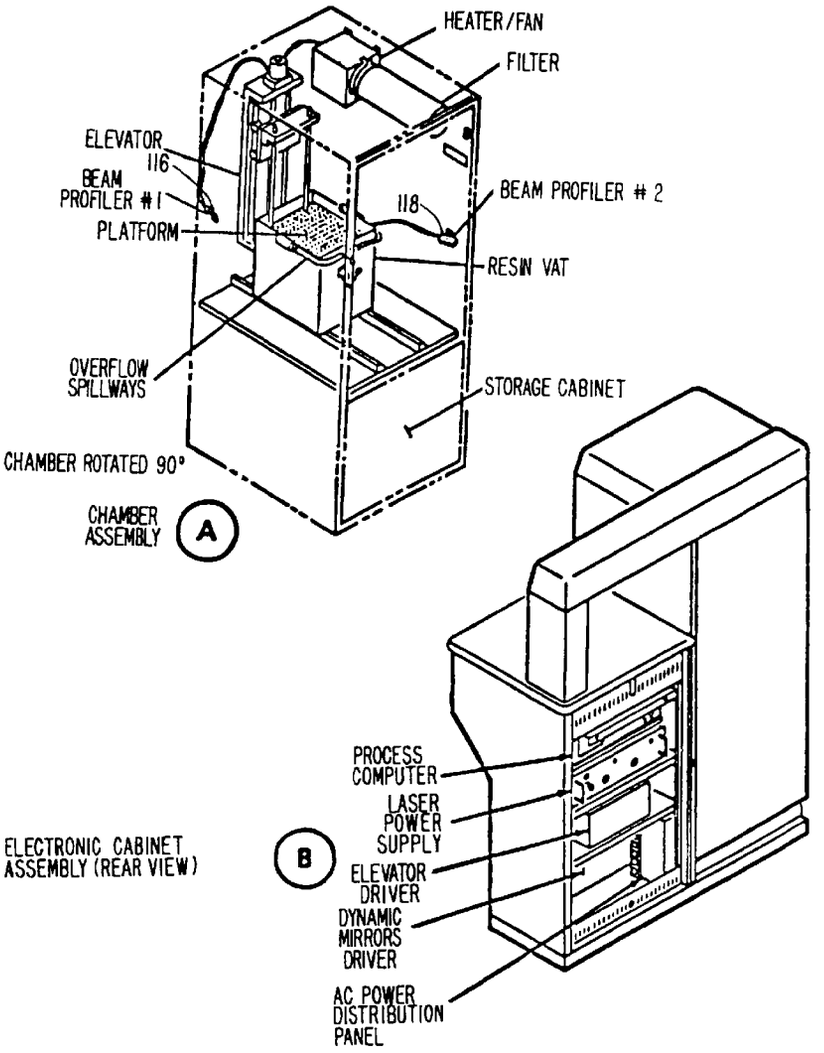

HEATER/FAN
FILTER
ELEVATOR
116
BEAM
PROFILER #1
PLATFORM
118
BEAM PROFILER # 2
RESIN VAT
OVERFLOW
SPILLWAYS
STORAGE CABINET
CHAMBER ROTATED 90°
CHAMBER
ASSEMBLY
A
PROCESS
COMPUTER
LASER
POWER
SUPPLY
ELECTRONIC CABINET
ASSEMBLY (REAR VIEW)
B
ELEVATOR
DRIVER
DYNAMIC
MIRRORS
DRIVER
AC POWER
DISTRIBUTION
PANEL

FIG. 7.

Description

Background of the Invention

1. Cross-Reference to Related Applications

This application relates to U.S. Patent Application Serial Nos. 182,823; 182,830; 183,016; 183,015; 182,801; 183,014; and 183,012; filed April 18, 1988, all of which are hereby fully incorporated into this disclosure by reference as though set forth in full herein. Continuations-in-part of U.S. Patent Application Serial Nos. 182,830, 183,016; 183,014; and 183,012; were filed on November 18, 1988, all of which are hereby fully incorporated into this disclosure by reference as though set forth in full herein. The Serial Nos. for the above-mentioned continuations-in-part are, respectively, 269,801; 268,806; 268,337; 268,907 (all for Ser. No. 182,830); 268,429 (all for Ser. No. 183,016); 268,408 (all for Ser. No. 183,014); and 268,428 (Ser. No. 183,012). A continuation application of U.S. Patent Application S.N. 269,801 was filed on March 31, 1989, which is hereby fully incorporated into this disclosure by reference as though set forth in full herein. The Lyon & Lyon Docket No. for the above-mentioned continuation application is 186/195.

2. Cross-Reference to Attached Appendices

The following appendices are affixed to this application, and are also hereby fully incorporated by reference into this disclosure as though set forth in full herein:

Appendix A: 3D Systems, Inc., Beta Site Users Manual and Service Manual, November, 1987
Appendix B: 3D Systems, Inc., Beta Release, First Draft, Software Manual, October, 1987
Appendix E: Non-3D Systems, Inc. Software Vendors as of April 13, 1988
Appendix F: Software Listing, Version 3.03
Appendix K: Software Listing, Version 3.20

3. Field of the Invention

This invention relates generally to apparatus and methods for measuring beams of electromagnetic radiation or particles and, more particularly, to a new apparatus and method for measuring the intensity profile of beams and related applications in connection with the production of three-dimensional objects by stereolithography.

4. Background of the Invention

In recent years, "stereolithography" systems, such as those described in U.S. Pat. No. 4,575,330 entitled "Apparatus For Production Of Three-dimensional Objects By stereolithography," have come into use. The disclosure of U.S. Pat. No. 4,575,330 is hereby incorporated by reference, as if fully set forth herein. Basically, stereolithography is a method for automatically building complex plastic parts by successively printing cross-sections of photocurable polymer or the like on top of each other until all of the thin layers are joined together to form a whole part. With this technology, the parts are literally grown in a vat of liquid plastic. This method of fabrication is extremely powerful for quickly reducing design ideas to physical form and for making prototypes.

Photocurable polymers ("photopolymers") change from liquid to solid in the presence of light and their photospeed with ultraviolet light (UV) is fast enough to make them practical model building materials. The material that is not polymerized when a part is made is still usable and remains in the vat as successive parts are made. An ultraviolet laser may be used which generates a small intense spot of UV light. This spot is moved across the liquid surface with a galvanometer X-Y mirror scanner. The scanner is driven by computer generated vectors or the like. Precise and complex patterns can be rapidly produced with this technique.

The laser scanner, the photopolymer vat, and an elevator, along with a controlling computer, combine together to form a stereolithography apparatus, referred to as an "SLA." An SLA is programmed to automatically make a plastic part by "drawing" one cross-section at a time, and building it up layer by layer.

Stereolithography represents an unprecedented way to quickly make complex or simple parts without tooling. Since this technology depends on using a computer to generate its crosssectional patterns, a natural data link to CAD/CAM exists.

To be effective, a stereolithography system must have information about the focus, laser beam oscillation mode, beam power, intensity distribution or profile, and scanning system drift of the drawing laser in order to carry out the accurate and efficient production of parts (objects made by stereolithography are known as "parts"). The beam must be in relative focus at the surface of the working photopolymer fluid. The laser mode, intensity distribution, and beam power

are important to the depth and width of cure of the working fluid, as well as the scan speed. The "drift" of the scanning system must be measured and corrected periodically.

Beam profile (a profile of the intensity of the beam) measurements provide useful information about the beam because they can help accomplish the following purposes: 1. Focus optics and correct astigmatism and other aberrations; 2. Measure the power of the beam (needed on a day to day basis); 3. Study the laser mode and changes of the mode; 4. Compensate for drift of the laser scanning system; 5. Allow recording of the drift for later analysis of changes; 6. Automatically calibrate the scanners; 7. Allow easy control of beam position for making other measurements (e.g. to independently measure the beam power in order to find the system's power calibration factor) and 8. Permit the prediction of the size and shape of the cured plastic trace.

Accordingly, a need exists for an apparatus and method to rapidly determine the profile of the beam, particularly in connection with a stereolithographic apparatus, to calibrate and normalize a stereolithographic apparatus, and to correct for the drift of the mirror positioning system of the stereolithographic apparatus.

Summary of the Invention

Briefly, and in general terms, the present invention provides a new and improved apparatus and method for profiling a beam. The apparatus comprises means for measuring the intensity of a portion of the beam when the beam is incident on the measuring means and means for changing the relative perpendicular displacement of the measuring means from an optical path followed by the beam in order to measure the intensity of some or all portions of the beam along a surface substantially perpendicular to the optical path followed by the beam. The method for profiling a beam comprises the step of measuring the intensity of a crosssectional portion of preselected size of the beam along a surface substantially perpendicular to the optical path followed by the beam and repeating the measuring step for other portions of the beam along the surface.

The apparatus and the method develops an intensity map of the beam along a surface substantially perpendicular to the optical path followed by the beam. The intensity map gives the intensities for each of the portions of preselected size of the cross-section of the beam. The intensity profile so gained can be used to determine and adjust the focus of the beam as well as to calculate the power of the beam (given a known power conversion factor). The profile of the beam may be used to predict the cure depth and width of plastic created on the working fluid. The beam profiling apparatus may be used to detect drift of the apparatus that scans the beam, by serving as a fixed reference point and determining the scanning apparatus coordinates of the center of the beam, which can be used to recalibrate the positioning "map" or table that directs the scanning apparatus in translating computer generated designs to actual dimensions on the surface of the fluid that solidifies to form the object.

The presently preferred version of the beam profiling system has a significant advantage of economy because it uses the computing system and light-beam positioning system which are already present in the stereolithographic apparatus. Although the present system refers to "laser beam" and "X-Y galvanometer scanning system" it is apparent that these advantages also apply to other possible systems with different energy sources or positioning means or combinations of these.

The present invention also provides a related new and improved apparatus and method for normalizing and calibrating a stereolithographic apparatus.

In an apparatus for the production of parts by stereolithography, it is useful to have an apparatus and a method for normalizing and calibrating the projection of the reaction means (in a preferred embodiment, a laser beam) upon a working medium (in a preferred embodiment, a designated surface of a photopolymer). Accurate positioning of the reaction means upon the working medium is made possible according to the present invention by providing at least one sensor capable of being positioned at a predetermined location in a plane defined by the designated surface of the working medium. The sensor is sensitive to the presence of the reaction means. Memory means are provided to store information which includes the positioning information which causes the reaction means to be pointed accurately at the sensor. In a preferred embodiment, a memory look-up table or map is prepared containing specific positioning information for each of a number of specific locations on the surface of the working medium. A standard linear interpolation technique is utilized to determine positioning information useful to point the reaction means at points intermediate to those in the look-up table.

Accordingly it is an object of this invention to provide an apparatus and method for accurately calibrating and normalizing a stereolithographic apparatus.

It is a further object of this invention to provide an improved and more accurate method and apparatus for pointing a beam.

The present invention also provides a related new and improved apparatus and method for correcting for drift of a mirror positioning system so as to provide improved accuracy and repeatability.

In a stereolithographic device, a three dimensional object or "part" may be produced from a medium capable of solidification upon exposure to a reaction means operating in a preselected manner upon a designated surface of the

medium. In this manner successive adjacent laminae are built up of solidified medium to form the part.

Thermal effects, wear effects and other effects can cause the positioning means for positioning the reaction means to "drift" so that the reaction means can no longer be repeatably directed to the same location. In order to alleviate such drift errors, the present invention utilizes at least one sensor means capable of detecting application of the reaction means to the sensor means, said sensor means being mounted in a predetermined location fixed with respect to the designated surface of the medium. Positioning means position and apply the reaction means and can apply the reaction means to said sensor means. Current sensor location output means provide current sensor location information indicating the location of the sensor means when the reaction means is applied to the sensor means. Likewise a memory means is utilized to contain past sensor location information which includes a prior apparent location of said sensor means. Means for comparing said current sensor location information and said past sensor location information is provided and is capable of determining a drift correction for use in adjusting the positioning means so as to eliminate drift error effects in the pointing of the reaction means.

The above and other objects and advantages of this invention will be apparent from the following more detailed description when taken in conjunction with the accompanying drawings of illustrative embodiments.

Brief Description of the Drawings

FIGS. 1, 2 and 3 are flow charts illustrating the basic concepts employed in practicing the method of stereolithography;

FIG. 4 is a combined block diagram, schematic and elevational sectional view of a stereolithography system;

FIG. 5 is a block diagram of a stereolithography system;

FIGS. 6 and 7 are exploded perspective views of the major component groups in a stereolithography system;

FIG. 8 is a perspective view of the laser and optical system in a stereolithography system which utilizes the preferred embodiment of the present invention;

FIG. 9A is a cross-sectional schematic view of a beam profiler sensor of the preferred embodiment of the present invention;

FIG. 9B is a top plan view of a pinhole plate for a beam profiler sensor of the preferred embodiment of the present invention;

FIG. 10 is a block diagram showing the apparatus of a preferred embodiment of the present invention;

FIG. 11A is a functional block diagram of the preferred embodiment of the method of generating an intensity profile of a beam according to the present invention;

FIG. 11B is a functional block diagram of a method of moving a beam in carrying out the method described in FIGURE 11A;

FIG. 11C is a functional block diagram of a method of reading the intensity of a portion of a beam in carrying out the method described in FIG. 11A;

FIG. 12 is a functional block diagram showing process and analyses which may be coupled with the method described in FIG. 11A;

FIG. 13 is a chart showing a sample intensity profile for a beam generated by a preferred embodiment of the invention;

FIG. 14 shows predicted cure depth profiles along two axes generated from beam profile information from a preferred embodiment of the present invention;

FIG. 15 depicts a cross-section of a cured photopolymer trace caused by exposure to a beam;

FIG. 16A depicts a perspective view of a calibration plate; and

FIG. 16B depicts a cross-sectional elevation of a calibration plate.

Description of the Preferred Embodiment

The stereolithographic system with which the apparatus and method of the preferred embodiment of the present invention is used generates three-dimensional objects by creating a crosssectional pattern of the object to be formed at a selected surface of a fluid medium, e.g., a UV curable liquid or the like, capable of altering its physical state in response to appropriate synergistic stimulation such as impinging radiation beams, or electron or other particle beam bombardment. Successive adjacent laminae, representing corresponding successive adjacent crosssections of the object, are automatically formed and integrated together to provide a step-wise laminar or thin layer buildup of the object, whereby a three-dimensional object is formed and drawn from a substantially planar or sheet-like surface of the fluid medium during the forming process. The technique is generally described in the flow charts and diagrams of FIGS. 1-5.

A stereolithographic system is shown in elevational crosssection in FIG. 4. A container 21 is filled with a UV curable liquid 22 or the like, to provide a designated working surface 23. A programmable source of ultraviolet light 26 or the like

produces a spot of ultraviolet light 27 in the plane of surface 23. The spot 27 is movable across the surface 23 by the motion of mirrors or other optical or mechanical elements (not shown in FIG. 4) used with the light source 26. The position of the spot 27 on surface 23 is controlled by a computer control system 28. The system 28 may be under control of CAD data produced by a generator 20 in a CAD design system or the like and directed in PHIGS format or its equivalent to a computerized conversion system 21 where information defining the object is specially processed to reduce stress, curl and distortion, and increase resolution, strength and accuracy of reproduction.

A movable elevator platform 29 inside container 21 can be moved up and down selectively, the position of the platform being controlled by the system 28. As the device operates, it produces a three-dimensional object 30 by step-wise buildup of integrated laminae such as 30a, 30b, 30c.

The surface of the UV curable liquid 22 is maintained at a constant level in the container 21, and the spot of UV light 27, or other suitable form of reactive stimulation, of sufficient intensity to cure the liquid and convert it to a solid material is moved across the working surface 23 in a programmed manner. As the liquid 22 cures and solid material forms, the elevator platform 29 that was initially just below surface 23 is moved down from the surface in a programmed manner by any suitable actuator. In this way, the solid material that was initially formed is taken below surface 23 and new liquid 22 flows across the surface 23. A portion of this new liquid is, in turn, converted to solid material by the programmed UV light spot 27, and the new material adhesively connects to the material below it. This process is continued until the entire three-dimensional object 30 is formed. The object 30 is then removed from the container 21, and the apparatus is ready to produce another object. Another object can then be produced, or some new object can be made by changing the program in the computer 28.

The light source 26 of a stereolithography system according to a preferred embodiment of the invention is typically a helium-cadmium ultraviolet laser such as the Model 4240-N HeCd Multimode Laser, made by Liconix of Sunnyvale, California.

A commercial stereolithography system will have additional components and subsystems besides those previously shown in connection with the schematically depicted systems of FIGS. 1-5. For example, the commercial system would also have a frame and housing, and a control panel. It should have means to shield the operator from excess UV and visible light, and it may also have means to allow viewing of the object 30 while it is being formed. Commercial units will provide safety means for controlling ozone and noxious fumes, as well as conventional high voltage safety protection and interlocks. Some commercial units will also have means to effectively shield the sensitive electronics from electronic noise sources.

The commercialized SLA is a self-contained system that interfaces directly with the user's CAD system. A commercialized SLA containing the preferred embodiment of the apparatus of the present invention, as shown in FIGS. 6 and 7, consists of four major component groups: the SLICE computer terminal, the electronic cabinet assembly, the optics assembly, and the chamber assembly. A block diagram of the commercialized SLA is shown in FIG. 5.

The electronic cabinet assembly includes the process computer, keyboard, monitor, power supplies, ac power distribution panel, and control panel. The computer assembly includes plug-in circuit boards for control of the terminal, high-speed scanner mirrors, and vertical (z-stage) elevator. Power supplies for the laser, dynamic mirrors, and elevator motor are mounted in the lower portion of the cabinet.

The control panel includes a power on switch/indicator, a chamber light switch/indicator, a laser on indicator, and a shutter open indicator.

Operation and maintenance parameters, including fault diagnostics and laser performance information, are also typically displayed on the monitor. Operation is controlled by keyboard entries. Work surfaces around the keyboard and computer are covered with formica or the like for easy cleaning and long wear.

Turning to FIG. 8, the helium cadmium (HeCd) laser 100 and optical components are mounted on top of the electronic cabinet and chamber assembly 102. The laser and optics plate may be accessed for service by removing separate covers. For safety reasons, a special tool is required to unlock the cover fasteners and interlock switches are activated when the covers are removed. The interlocks activate a solenoid-controlled shutter to block the laser beam when either cover is removed.

As shown in FIG. 8, the shutter assembly 104, two ninety degree beam-turning mirrors 106, 108, a beam expander 110, an X-Y scanning mirror assembly 112, and precision optical window 114 are mounted on the optics plate. The rotary solenoid-actuated shutters are installed at the laser output and rotate to block the beam when a safety interlock is opened. The ninety degree beam-turning mirrors 106, 108 reflect the laser beam to the next optical component. The beam expander 110 enlarges and focuses the laser beam on the liquid surface. The high speed scanning mirrors direct the laser beam to trace vectors on the resin surface. The 2- mirror, 2-axis galvanometer scan heads sold by General Scanning Inc. of Watertown, Massachusetts have been found to be satisfactory for this purpose and in a preferred embodiment their Model DX-2005 servo and Model XY-0507 galvanometer X-Y scanning heads are used. A quartz window 114 between the optics enclosure and reaction chamber allows the laser beam to pass into the reaction chamber, but otherwise isolates the two regions.

The chamber assembly contains an environmentally controlled chamber, which houses a platform, resin vat, ele-

vator, and beam profilers.

The chamber in which the object is formed is designed for operator security and to ensure uniform operating conditions. The chamber may be heated to approximately 40°C (104°F) and the air is circulated and filtered. An overhead light illuminates the reaction vat and work surfaces. An interlock on the access door activates a shutter to block the laser beam when opened.

The resin vat is designed to minimize handling of the resin. It is typically installed in the chamber on guides which align it with the elevator and platform.

The object is formed on a platform attached to the vertical axis elevator, or Z-stage. The platform is immersed in the resin vat and it is adjusted incrementally downward while the object is being formed. To remove the formed part, it is raised to a position above the vat. The platform is then disconnected from the elevator and removed from the chamber for post processing. Handling trays are usually provided to catch dripping resin.

Two beam profiler sensors 116, 118 according to the preferred embodiment of the present invention are mounted at the sides of the resin vat and the focal point of the laser optical system is adjusted to match radial sensor positions (i.e., they are mounted at a radial distance from the galvanometer scanners equal to the distance from the galvanometers to a point 7.62 mm (0.3 inches) below the surface of the liquid). (See FIG. 7). The scanning mirror is periodically commanded to direct the laser beam onto the beam profiler sensors, which measure the beam intensity profile. The data may be displayed on the terminal, either as a profile with representation of intensity values or as a single number representing the overall (integrated) beam intensity. This information is used to determine whether the mirrors should be cleaned and aligned, whether the laser should be serviced, whether the scanner mirrors have drifted, and what parameter values will yield cured vectors of the desired thickness and width.

The beam profiler sensors 116, 118 are symmetrically placed relative to the center of the resin vat. (See FIG. 7). They should preferably have similar X and Y offsets as measured from the vat center (of opposite values) although this is not required; that is to say, they are on a diagonal of the stereolithographic apparatus. In FIG. 7, the beam profiler sensors 116, 118 are seen in the corners of the chamber assembly. The distance from the second scanning mirror on the optical plate above the chamber assembly to each beam profiler sensor aperture is the focal length which equals the desired liquid to scanning mirror length plus a small increment. In the SLA-I sold by 3D Systems, Inc. (which is shown in FIGS. 6 and 7), this liquid to scanning mirror length is approximately 68,6 cm (27 inches) and the small increment is an additional 7,62 cm (0.3 inches), therefore the focal length is approximately 69,36 cm (27.3"). The distancing of the beam profiler sensors 116, 118 from the second mirror by the desired focal length has the effect of detecting the best average focus for the designated surface of the photopolymer in the SLA-I resin vat. At the center of the resin vat, when the photopolymer is at the desired level, the focal length of the laser beam will be 9,62 mm (0.3 inches) below the surface of the photopolymer. The focus of the beam at the surface of the photopolymer in the center of the resin vat will not vary much. At the corner of the 30,48 cm (12 inch) resin vat of the SLA-I, the focal length will be approximately 2,54 cm (1 inch) above the surface of the photopolymer. The focal length will be at the surface of the photopolymer at a circle with a 10,67 cm (4.2 inch) radius about the center of the surface of the photopolymer. The placement of the beam profiler sensor at the focal length is intended to obtain an optimal beam profile considering that the surface of the photopolymer will mostly not be at the focal length of the laser.

FIG. 9A is a cross-sectional view of a beam profiler sensor 35 of a preferred embodiment of the apparatus of the present invention and FIG. 9B is a top plan view of a pinhole plate used in the beam profiler sensor. The beam profiler sensor has a thin stainless steel metal plate 40 having four etched pinholes 45 of varying size. In a preferred embodiment these holes have diameters 0,0127 mm (0.0005"), 0,0254 mm (0.001"), 0,0508 mm (0.002"), and 0,1016 mm (0.004"). The pinholes each permit a small fraction of the laser beam 50 incident upon the pinhole to fall on a photodetector 55 underneath the plate 40. The purpose for providing several pinholes is to permit profiling of beams having a wide range of incident power. One of the pinholes will be best suited for measuring the intensity profile of a beam of a given incident power. For the HeCd lasers used in the SLA-I, a pinhole of 0,0508 mm (2 mil) or (0.002 inch) diameter has been found to be satisfactory. The beam is scanned across a selected pinhole in an X Y array to build up a two dimensional profile of the beam intensity.

As may be seen in FIGS. 7 and especially 9A, the beam profiler sensor 35 has a two part housing 60. Light beam 50 enters from the right and moves toward the left in FIG. 9A. The beam profiler sensor is mounted in the corners of the chamber assembly compartment in such a way as to prevent the resin vat from hitting the beam profiler sensor when it is moved into and out of the compartment (see FIG. 7).

Turning to FIG. 9A the beam profiler sensor 35 comprises a split two part housing 60, pinhole plate 40, ultra-violet transmitting filter 70 that absorbs visible light and prevents spurious readings due to visible light. Filter 70 is a two millimeter thickness of Schott UG-11 filter glass which has been found to be acceptable for this purpose in a preferred embodiment. The characteristics of this filter provide reasonable transmission of light in the 300-370 nanometer wavelength region with considerably less transmittance at other wavelengths. A one millimeter thickness of HOYA U-350 filter material would also be acceptable.

Underneath the filter 70 in the beam profiler housing is a photodiode sensor 55 which detects the ultraviolet light

which passes through the filter 70 from the pinhole 45. An EEG Vactec VTS 3072 super blue enhanced photodiode has been found to be acceptable. The output from this photodiode is passed to a current to voltage amplifier (not shown). An OP07 current to voltage amplifier whose implementation is well known to those skilled in the art has been found to be acceptable.

The pinhole plate 40 of the beam profiler sensor 35 is covered with a quartz filter (not shown). The quartz filter is cleanable and protects the beam profiler sensor from dust and photopolymer drips. The quartz filter should be coated to prevent internal reflections when the photosensor is not perpendicular to the beam, in order to prevent false shape measurements. Optionally a diffuser (not shown) can be used between the pinholes to aid the filter to protect the optical components from damage.

FIG. 10 is a block diagram showing the apparatus of the preferred embodiment of the present invention. Fundamental to the invention is a control and analysis computer. This computer receives inputs from a program, a keyboard or the like and may display results through a printer or terminal or the like. The control and analysis computer sends positioning commands to a mirror positioning system which controls the X Y scanner mirrors. The laser beam is focused by the optics shown in FIG. 8 to reach the X Y scanner mirrors and is directed by those mirrors to one of the beam profiler sensors. The use of two beam profiler sensors is recommended for the purpose of drift correction. The sensor signal from the beam profiler sensors is converted to a signal readable by the computer which is then sent back to the control and analysis computer to be manipulated as described hereafter.

In physical terms, the beam profile method according to the present invention causes the beam to be moved to each of the points of an array on the pinhole plate centered on the best known position of the pinhole. As a result, different sectors of the beam will fall on the pinhole and will be transmitted through the same to be detected by the photodiode and converted into a numerical signal that can be analyzed by the computer. A profile of the intensity of different sectors of the beam will be built up by the computer (see FIG. 13). This is the "intensity profile" of the beam.

FIG. 11A is a functional block diagram showing how a beam profile is developed according to the preferred embodiment of the invention. The best known location of a pinhole on the beam profiler sensor is called up from memory by the control and analysis computer and sent to the mirror positioning system to position the X Y scanner mirrors to direct the beam at this best known location. The control and analysis computer, through the beam positioning system, moves the beam to the first row in the first column of a square array centered on the best known location. The intensity of the portion of the beam entering the pinhole as detected by the beam profiler sensor is then read and is saved as well as the mirror position commands which are associated with that intensity. The beam is then moved in sequence from the first to the last array points on a particular row or column and the reading and saving intensity value steps are repeated. The beam is then moved in sequence from the first to the last array columns or rows and the moving and reading steps are then performed down each column or row. The result is that beam intensity readings are taken for each of the positions on the array (a "position" is known to the computer as a set of mirror positioning commands). A standard analysis, performed by the control and analysis computer of the array of intensity values is usually performed to generate a new best known location of the pinhole (for use in performing the first step of the scan profile the next time) regardless of the detailed function actually being analyzed (see FIG. 12). This calculated best known position can be found with great precision by this means, to an accuracy much finer than the size of the pinhole. When an appropriate number of locations have been found and stored, the control system can use these values with two-axis linear interpolation to move the beam to those or any intermediate locations.

FIG. 11B is a functional block diagram of a method of moving a beam and carrying out the method described in connection with FIG. 10A. To move the beam, the first step is to send the beam positioning information to the servomechanism of the X Y scanner mirrors concerning the desired location. The servomechanism (which may be analog or digital) then send a signal to the mirror drivers to position the X Y scanner mirrors to a new location. The servomechanism of the X Y scanner mirrors measures the actual position of the mirror drivers and compares the actual position with the intended position and adjusts the drive signals accordingly. The adjustments are continued within specification values of the intended location.

FIG. 11C is a functional block diagram of a method of reading the intensity of a portion of a beam and carrying out the method of the preferred embodiment of the present invention. The first step is to convert the total amount of light which passes through the pinhole into a signal which is proportional to that amount of light. In the preferred embodiment, this process is performed by the photodiode which measures the light coming through the pinhole and the filter. The current from the photodiode is sent to a current to voltage amplifier which generates a signal proportional to the amount of light received by the photodiode. Amplification of the signal (which will be proportional to intensity) to provide wide dynamic range of the measurement is important to obtain small but significant readings for the edge of the beam which would otherwise be lost. The next step is to measure the signal, which is proportional to the amount of light received, after the signal is converted into a digital form for numerical analysis.

FIG. 12 is a functional block diagram showing the processes and analyses which may be coupled with the method described in FIG. 11A. As may be observed from that figure, a number of different processes and analyses may be selected from a menu, the first five of which are connected to the scan profile routine of FIG. 11A. The first step is to

scan the intensity profile of the beam according to the method described in connection with FIG. 11A. The intensity profile may be displayed numerically or in the form of a graph. As an option, the power may be calculated from the intensity profile as well as a new best known location of the pinhole used. Another possible process is to add the data generated in connection with the beam intensity profile to a history file with an option of displaying the history file. A further possible process is to calculate and display drift information for the mirror positioning system which generally involves scanning a second separate sensor (in the case of the preferred embodiment, another beam profiler sensor) and then calculating and displaying the offset and gain terms of the drift. Another process is to calculate and display the power of the beam, which involves summing up the intensities of a profile and multiplying by a power conversion factor. The power conversion factor can be determined, for example, by utilizing the process with a beam of known power or by comparing calculated power to that of a calibrated sensor and determining the required gain factor. A further function is to calculate and display focus information, with a possible option of using a special transformation of the intensity data used to calculate focus information and the use of known resin properties to predict the shape and size of the cured traces of photopolymer. Another possible function is to move the beam to the desired location for set up (to make parts), tests and so forth with an option for searching for the sensor or scan profile from this new location. A useful function is to search for sensor pinholes in a spiral move-read format. This may be necessary if the best known location of the pinhole is not accurate, in case when the array is traced over the best known location of the pinhole is not detected. A further step may be to verify findings (of pinhole location) by test or scan profile. A still further function is the use of the beam profilers in calibration, which involves the measurement of the drift while obtaining a calibration map for the surface corresponding to the surface of the photopolymer. A final function is to store in the machine the parameters of best known location, scale factors, resin properties, and so forth.

FIG. 13 is a chart showing a sample intensity profile for a laser beam as generated by a preferred embodiment of the present invention. The numerical values correspond to the measured intensity of a beam from a preferred embodiment of the present invention. The numbers have been converted to integers for ease of reading the display.

The intensity profile generated by the present invention may be used to calculate the power of the beam and to predict the shape and dimension of a cured trace of photopolymer (the solidified photopolymer due to exposure to the beam of UV light), as the following discussion will demonstrate.

The intensity of the beam is measured by the beam profiling apparatus when the beam is directed at each point of an array on the pinhole plate, which is a surface generally perpendicular to the optical path taken by the beam. The X and Y directions on this surface correspond to the directions taken by the beam when one or the other of the scanning mirrors rotate.

The X and Y array coordinates are 1 to i_{max} and 1 to j_{max}, respectively (typically i_{max} and j_{max} are each equal to 22).

The beam is typically stepped or moved from point to point in the array, the time for the movement being much less than the time the beam remains at a point. The distance between points is:

$$s\ (mm) = scanstep / scalefactor \qquad [Eq.\ 1]$$

The scanstep is typically 4 "bits" and the scale factor is normally 140 bits/mm. The scanning mirrors can each take on 65535 (64K) different positions over a 40° optical beam rotation. This in turn means that 1 bit along the X or Y axis corresponds to a rotation of 6.104×10^{-4} degrees. Since mirror to liquid distance is approximately 27" this angular rotation corresponds to a translation at the liquid surface of 2.875×10^{-4} inches or equivalently, 127 bits/mm or approximately 140 bits/mm.

The array area must cover the full beam (measurement of the entire beam is needed for power calibration as well as to produce the maximum information concerning the beam), and must have a number of points sufficient to resolve the desired beam profile. Typically, the spacing between points in this array is less than one tenth of the beam width. The pinhole diameter should be smaller than this resolution limit.

An "element" is the portion of a beam measured when the beam is directed at a point (m,n) of the array. Each element (m,n) has an intensity reading I(m,n). The letters m,n refer to positions or points in the X, Y directions in the array, respectively. FIG. 13 shows intensity readings in an array much as discussed here.

The beam power is measured independently and the power calibration factor k is derived from the following equation:

$$k * \sum_{m,n} I(m,n) = Power\ (watts) \qquad [Eq.\ 2]$$

The power calibration factor k only applies to the selected pinhole and measuring system and laser wavelength. The independent power measurement must be made on the beam after the beam has traversed the same number of optical

surfaces in the beam path. These calculations also assume that the background light signals have been eliminated and amplifier scaling compensated for.

The power/unit area at element (m,n) is given by:

$$\text{Intensity (at element m,n)} = k * I(m,n) / s^2 \text{ (watts/mm}^2\text{)} \quad \text{[Eq. 3]}$$

This is the instantaneous intensity experienced by a small area in the (m,n) element, regardless of whether the beam is stationary or moving.

When the beam is moving uniformly at the speed v (mm/sec) along the Y axis, then each element takes a time equal to s/v to pass, and the exposure energy absorbed per unit area from the element (m,n) is:

$$\text{Exposure from element (m,n)} = k * I(m,n) / s^2). (s / v) \text{ (Joules/mm}^2\text{)} \quad \text{[Eg. 4]}$$

This is the energy absorbed per unit area from a particular element (m,n) of the beam.

The total exposure (beam energy absorbed) as the entire beam passes an area equivalent in size to an element, as defined above, is:

$$\text{Exposure at m} = k * \sum_n I(m,n) / s^2) * (s / v) \text{ (Joules/mm}^2\text{)} \quad \text{[Eg. 5]}$$

Physically speaking, this equation represents a situation in which the beam traverses in the Y direction over an area equivalent in size to an element of the beam as the term element is used above. The area is traversed by the elements of the beam corresponding to the X coordinate m, so the element-sized area at m is exposed to all elements (m,n) of the beam as n varies between 0 and j_{max}.

The calculations described above are based on discrete elements, but could obviously be generalized to use integrals. The movement is assumed along the Y axis for convenience. Other angles can be simply derived, and may be necessary if the beam is asymmetrical.

The speed v corresponds to parameters SS and SP as follows:

$$v = (SS / \text{ScaleFactor}) (SP / 100,000) \text{ (mm/sec)} \quad \text{[Eq. 6]}$$

Where:

SS	= Step Size in bits/step;
ScaleFactor	normally is 140 bits/mm;
SP/100,000	= Step Period in seconds (SP units are units equal to 10 microsec); and
1E6	= 1,000,000 is a conversion factor between Joules / mm^2 and Joules / m^2 or microJoules / mm^2

Equations 5 and 6 are combined to calculate the total exposure (power incident, i.e., energy) at the sensor or at the liquid (photopolymer) surface represented by Z=0 at a small area at position in as the beam is moved in the Y direction: Exposure at (m,z=0):

$$E(m,z=0) = \sum_n I(m,n) * \frac{k * SP * \text{ScaleFactor} * 1E6}{s * SS * 100,000} \quad \text{[Eg. 7a]}$$

$$= \frac{\sum_n I(m,n) * SP * \text{ScaleFactor} * \text{power} * 10}{\sum_{m,n} I(m,n) * SS * s} \text{ uJ / mm}^2 \quad \text{[Eg. 7b]}$$

Finally, the absorption of the beam as it penetrates into the liquid may now be compensated for in accordance with Beer's law:

$$E(m,z) = E(m,0) * \exp(-z/\text{lambda}) \quad \text{[Eq. 8]}$$

Where:
lambda is the penetration depth (mm);
E(m,0) is the summed exposure at the surface; and
E(m,z) is the exposure at depth z below the surface (mm).
Attenuation is assumed to have no nonlinearities or time-dependence in absorption, being represented simply by I(z)=I(z=o) * exp (-z/lambda) . It is clear that appropriate modifications can be made to the foregoing calculations to allow for deviations from the above simple absorptive behavior.

The photopolymer has been shown experimentally to cure to a gel if the exposure is greater than a critical value Ec, so for any given system the shape of a trace of cured plastic can be predicted by calculating the locus of points having exposure Ec. Ec can be accurately and separately measured for each photopolymer. The "gel point" gives only the "cured" versus "not cured" boundary, and disregards the gradient in exposure (related to penetration depth) at resin depths other than the Ec boundary depth. Part strength seems to relate to the higher exposure, and so absorption characteristics should be chosen to give the best (highest) cure gradient. The gradient or penetration depth also limits the best available resolution in the Z direction, because some variation in exposure (crossing lines, etc.) is unavoidable and this results in cure depth changes with this variation in exposure.

For any X location (m) the cured depth $z_c(m)$ is found from:

$$z_c(m) = \text{lambda} * 1n\,(\,E(m,z=0) / Ec\,) \quad \text{[Eq. 9]}$$

A beam profile measured with sufficient reliability and precision can be used to predict cure depths dependent only on chemical properties of a resin. FIG. 14 shows two examples of such predictions, along the X and Y axis respectively. The profile function (m,z) also permits prediction of the trace width as a function of depth (and with appropriate modifications, "beam width" and "minimum surface angle") automatically. Making and measuring "banjo tops," i.e., traces cured by the beam to directly determine the shape and dimensions of the cured photopolymer, will only be needed as an internal check of the system. FIG. 15 shows a cross-section of a test trace from a banjo top. FIG. 15 should be compared with FIG. 14.

To display the predicted trace profile, a scaled diagram of cure depth versus position is plotted across the beam. The scale factor for distance across the beam is easy, where one simply selects one column (or row, etc.) of the scan, with dimension s, to correspond to a pixel or graphics display block. The depth scale is then lamoda/s pixels (or blocks) for each increase in exposure by a factor of e. The only arbitrary feature is the zero of depth, related to the characteristic exposure Ec of Eg. 9, or equivalent factors from Eq. 7. The useful depth to be displayed is determined by the dynamic range of the intensity measuring system, and truncation at I(m,n)≥ any appropriate value close to the noise level.

Software code listings of the software of a preferred embodiment of the present invention are attached as Appendices F and K to the subject applications.

Calibration and Normalization

As pointed out above, it is desirable in a stereolithographic apparatus to provide an apparatus and method for calibrating the pointing of a reaction means on a working medium for improved precision and accuracy.

Calibration procedures of a preferred embodiment of the present invention allow the creation of a "map" from a design in CAD space to drawing instructions on the real SLA. In any automated building system there will be several different sources of error needing to be corrected by calibration procedures. The present system has a pair of scanning mirrors close together, and, if left uncorrected a simple mapping of CAD dimensions to the angles of the mirrors will result in pincushion distortion. This is because the system is building on a flat surface where further from the point on the surface nearest to the mirrors, equal increments of angle will project to progressively greater distances on the surface. This happens to be the dominant distortion in the present system, and is predictable from geometry so that its correction could be calculated. However there are numerous other errors and distortions which need to be compensated, and many of them are not readily predictable.

The calibration and normalization system of the present invention is useable in a broad range of applications and systems and automatically generates a "look-up table" to enable the conversion of CAD locations into instructions which are to be sent to the scanning system so that the intended pattern will be drawn on the working surface. The term "Normalization" may be used to indicate that more than one dimension at a time is being corrected, while "Calibration" could have connotations of applying a single scale factor to a system. In a preferred embodiment, the apparatus has a single position (beam profile) sensor which is moved automatically to an array of points on the working surface, and then a record is made of the corresponding mirror instructions needed to reach each of these points. In another preferred embodiment, a square array of sensor pinholes is utilized so that no movement of the sensor was required. In yet

another preferred embodiment, a linear array of pinhole sensors needing to be moved along only one axis is utilized.

Turning to FIGS. 16A and 16B the square calibration plate 200 of a preferred embodiment of the present invention is depicted. An ultraviolet-opaque metallic coating 206 is provided by evaporation on a substrate 204 made preferably of quartz or pyrex of 3,175 mm (1/8-inch) to 6,35 mm (1/4-inch) thickness. In a preferred embodiment an array of 49 by 49 holes 202 are etched in the UV-opaque metallic coating 206 on 6,35 mm 1/4-inch centers. Each etched hole has a diameter of 0,102 ± 0,013 mm (0.004" ± 0.0005") although it is only important that the hole be of smaller diameter than the diameter of the projected beam at the plate for best resolution. Sensors (not shown) are mounted below the plate and are arranged when the calibration plate is in use to be in the precise location of the surface of the working medium.

In a presently preferred embodiment of the invention, an array of 5 x 5 or 25 photodiodes 208 sensitive to UV light are utilized in conjunction with the calibration plate. Because UV light can only enter the plate through one of the pinholes and because the plate material tends to diffuse light that enters, light entering a pinhole will travel horizontally beyond the exact location of the pinhole so that in a preferred embodiment 25 sensors such as the sensors herein previously described are adequate to cover the entire 49 x 49 array of pinholes.

Typically a calibration procedure will be performed prior to shipping an SLA to a customer and after any physical trauma to the mirror control system which would tend to uncalibrate the SLA.

In operation the calibration procedure uses the same beam profiler methodology to obtain a "best location" of a pinhole from the centroid of the scan data.

A new "best location" is obtained for each pinhole in the case of the plate, each pinhole-column location in the case of the linear array, and each predetermined location in the case of a sensor positioned at certain predetermined locations. It is not always necessary to scan each pinhole in order to obtain a working and practically useable look-up table. In a preferred embodiment of the present invention only about 40 x 40 pinholes are located and mapped. In situations where there are smaller geometric distortions, or less precision desired, or the other sources of distortion are suitable for correction with more reliance on interpolation, fewer pinholes could be mapped. Linear interpolation relative to the X and Y locations is used to determine mirror positioning for points intermediate the "best locations" stored in the look-up table in memory. The appropriate number of pinholes will be determined from these considerations, and from the time needed to make the calibration, and from the system memory available for storing the look-up table.

Optionally and preferably the drift correction apparatus and method described farther on are used in conjunction with calibration to obtain more precise and more accurate results. Similarly, during part building the same drift correction method and apparatus are used to improve precision and accuracy.

Briefly the method of a preferred embodiment of the present invention is set forth below:

STEP 1: User inserts calibration plate into SLA building chamber with its sensor pinholes positioned where the liquid surface normally stays; user specifies the delay times between reading sensors 1 and 2 (Fixed Beam Profile Sensors 116, 118) and the Calibration Plate (deemed "Sensor 3") locations.

STEP 2: Sensors 1 and 2 are located again as they are located when the first scan profile occurs to finalize and store their apparent coordinates in terms of mirror position information.

STEP 3: Center the calibration plate by determining if the center sensor 3 leading coordinates are within acceptable tolerances. The center plate coordinates should correspond to the mirror positioning center coordinates. This configuration allows maximum equal mirror movements in all directions.

STEP 4: The gain of the calibration plate (Sensor 3) is set by reading the center hole of the plate (which is found with the find sensor algorithm defined in Beam Profile Fig. 1). The actual beam intensity is derived by reading Sensor 3 with and without the beam in the center hole. This subtracts the background noise Sensor 3 receives. The gain control is adjusted by the user until the sensor sensitivity is optimized.

STEP 5: The borders of the calibration plate are established by stepping the beam from plate holes to the borders of the plate (1 = West, 2 = North, 3 = South, and 4 = East).

A) Move in direction 1, locating holes along the way by moving a predetermined (bits/hole separation) value. Bits refer to mirror coordinate change values.

B) When the known number of holes before the border are read, one more "bits/hole separation" movement is executed.

C) If a read finds a hole there, either the gain is set incorrectly because a false hole has been read, or the hole to the right of the center hole was where the search started. Return to Step 3.

D) If no hole is detected, the left and right borders are now established.

E) The beam returns to the center hole and begins a search for the back border in a similar fashion to A-D.

F) Once all borders are established through movements 1 and 2, the values of the hole locations for movements 1, 2, 3, and 4 are used to build a "rough map of hole separations" across the plate in mirror bits. Movement 4 leaves the beam at plate hole (1,1).

STEP 6: Quick search for all plate hole locations. If any hole cannot be located the beam is left at its most probable location, as determined by the "rough map of hole separations", and prompts the user to examine for dust around the hole location. After the wait the find is resumed until the hole is found or the user aborts. The gain may be reset in this process if the user desires. If the gain is changed the user restarts the quick search at hole (1,1).

STEP 7: After the quick search locates all the plate holes a final search is performed with the requested delays from Step 1. Also, on both quick and final searches the locations of sensors 1 and 2 are found to determine GAIN and OFFSET corrections to be made at the intervals (end of each row) of mirror bit movements to the same locations. These correction factors are applied proportionately to correct each of the calibration locations in a manner normalized to a single set of reference locations of the fixed sensors 1 and 2.

STEP 8: After the final search stores all the gain, offset, intensity, and locations date the procedure ends.

Drift Correction

Drift correction is a procedure which compensates for drift of, inter alia, the mirror positioning system by periodically checking on the apparent position of one or more beam profile sensors (herein referred to as "sensor means"). Measurements of changes in the apparent position of a single sensor allow for compensation for drift of the "zero setting" of the mirror system, two separate sensors allow additionally for correcting for otherwise uncompensated gain in the system and/or otherwise uncompensated changes in size of parts of the SLA due to thermal and other effects. Other errors can be corrected by utilization of even more sensors although in a preferred embodiment of the present invention two beam profile sensors are deemed sufficient.

In a preferred embodiment of the present invention, a calibration operation is periodically performed. In the calibration procedure, a plate having a number of holes and sensors is utilized to generate in system memory a table of mirror position settings which correspond to fixed predetermined locations on the plate.

During the time a calibration is being run, the system periodically checks on the apparent locations of two sensors. These measurements are used to correct the calibration measurements for this amount of drift, so that the values are all normalized to a "standard" pair of apparent positions of the two sensors. When later building a part, the same two sensors are again scanned periodically, and the apparent positions can be used to correct for changes in the zero and gain of the mirror system relative to the time at which the calibration was made. This procedure has been found to remove 90% of the error caused by drift of the mirrors. A description of the calibration procedure is provided earlier.

In the drift compensation method and apparatus of a preferred embodiment of the present invention two beam profile sensors capable of detecting when the laser beam is directed toward them by the mirror positioning system are mounted fixedly in predetermined locations fixed with respect to a designated surface of the solidifiable working medium upon which the reaction means impinges to solidify said medium.

Periodically the laser beam is directed toward the sensors, and sensor location output means provide a readout of the apparent location of the sensors. The current apparent location of the sensors are compared with a past apparent location of the sensors which is stored in memory and a difference indicates the need for a drift correction.

For example, when a single sensor No. 1 is in use, this sensor No. 1 had a past apparent position of X = 20, Y = 20 and a current apparent position of X = 22, Y = 22 then a drift of +2X and +2Y has occurred and the mirror positioning system can apply an appropriate correction factor in order to point the beam in the desired location. As another example, when in addition to this first sensor, a second sensor is used, then the second sensor might be read at calibration to be X = 64000, Y = 64000 and have a current apparent position of X = 64004, Y = 64003. Were this the case, in addition to a linear shift of +2X, +2Y across the entire system (the two sensors conveniently being located on a diagonal) there would also be a gain or stretching of +2X, +1Y of the apparent distance between sensors No. 1 and No. 2 and we expect and correct for proportionately different stretching for different locations relative to the sensor No. 1. In compensating, linear interpolation could be used to help the mirror positioning system compensate for the gain term in the drift error.

To achieve more accurate part-building using stereolithography, a special computer software algorithm was developed. This algorithm, known as the Drift Correction algorithm, can be considered having two parts.

The first part is code that determines the value of the variable used in drift correction. The second part is code for the application of these variables to correct for drift.

The drift correction variables are:

* the X-axis gain called DriftGainX in our examples
* the X-axis offset DriftOffsetX
* the Y-axis gain DriftGainY
* the Y-axis offset DriftOffsetY

The variables are used to compute a correction factor that is applied to all X,Y coordinate positions of the laser mirrors. This correction is applied to each axis separately and has a gain (multiplier) and offset (addition) term for each axis. It is in the form:

corrected axis value = (gain * desired axis distance from first sensor) + offset of first sensor

where the gain and offset variables map the ideal desired axis position to the corrected real-world position based on measurements.

The drift correction variables are determined at the start of each layer through the use of Beam Profiling algorithms very similar to those found in the Beam prograin. The position of two beam profile sensors at opposite corners of the vat is measured and calculated and compared to the ideal positions remembered in a disk data file. The differences between these two sets of profile sensors positions determines the value of the drift correction variables. Placement of the two profile sensors at the corners of the scanning field optimizes the baseline along both X and Y axes from which to measure changes in gain.

The Pascal function below is a sample for computing the values of the drift correction variables. The 'Ref' variables are the reference positions of the two beam profile sensors read from the disk data file. The 'Now' variables are the most recently determined positions of the two profile sensors. Because of dynamic changes in the physical characteristics of the laser scanning system, the positions held in the 'Now' variables will differ slightly from the ideal reference positions. The goal is to use these position differences to adjust upcoming mirror positions.

```
{
     CorrectForDrift -- compute value of drift correction
variables
}
procedure CorrectForDrift;
var
     quotient: Float Type;
begin
     DriftGainx     := (Sensor2_NowX - Sensor1_NowX) /
                       (Sensor2_RefX - Sensor1_RefX;)
     DriftOffsetX   :=  Sensor1_NowX;
     DriftgainY        (Sensor2_NowY - Sensor1_NowY)/
                       (Sensor2_RefY - Sensor1_RefY;)
     DriftOffsetY   :=  Sensor1_NowY;
end;
```

The Pascal procedure below shows how a coordinate pair can be corrected for drift once the drift correction variables have been determined. Note that the correction for each axis is separate from the other.

```
{
     DriftCorrect   -- correct coordinate pair for drift
}
procedure DriftCorrect(oldX,oldY: FloatType;var newX,newY:
FloatType);
begin
     newX :=  (oldX  -  Sensor1_RefX)  *  DriftGainX  +
DriftOffsetX;  newY := (oldY - Sensor1_RefY) * DriftgainY
+ DriftOffsetY;
end;
```

The DriftCorrect procedure is used to correct the start and the end of each drawing vector before it is converted to a "feature" which is a highly optimized form of a laser drawing used within 3D Systems software. The CorrectForDrift is used only once per layer, after the new position of the two beam profilers has been determined.

In practice, these routines are located within a special memory resident driver (known as STEREO) that different stereolithography application programs can use. The application program is responsible for finding the profiler location, and it passes this information along to the special device driver.

BUILD		STEREO
an applications program used to build parts; it passes along profile position info for each layer	-->	the memory-resident driver that performs several Stereolithography related tasks, one of which is drift correction

An equivalent drift correction procedure is performed on all of the calibration points during the calibration of the SLA. In a presently preferred embodiment, drift is determined at the end of a scan of each row of predetermined fixed positions and the correction is applied through linear interpolation for that row. A method of actual software implementation of such a routine for compensation would be well within the capabilities of one of ordinary skill in the art and therefore will not be further described herein although actual software for drift correction in an actual embodiment of the SLA-I is included in attached Appendices F and K.

It will be apparent from the foregoing that, while particular forms of the invention have been illustrated and described, various modifications can be made without departing from the spirit and scope of the invention. Accordingly, it is not intended that the invention be limited, except as by the appended claims.

Please now refer to the attached annexes which form part of the description.

APPENDIX A

© 1989 3D SYSTEMS, INC.
ALL RIGHTS RESERVED



systems, inc.

SLA-1

BETA SITE

STEREOLITHOGRAPHY

SYSTEM

CONFIDENTIAL LEGENDS ON THIS
APPENDIX ARE TO BE CONSIDERED DELETED.

3D Systems Incorporated
12847 Arroyo Street
Sylmar, California 91342
(818) 898-1533 • FAX (818) 361-5484

NOVEMBER 1987



systems, inc.

NOT TO BE DISCLOSED

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION RELATING TO STEREOLITHOGRAPHY PROCESSES AND EQUIPMENT WHICH ARE THE EXCLUSIVE PROPERTY OF 3D SYSTEMS, INC. THIS INFORMATION SHALL NOT BE DUPLICATED, USED OR DISCLOSED -- IN WHOLE OR IN PART -- FOR ANY PURPOSE OTHER THAN TRAINING AS AUTHORIZED IN WRITING BY 3D SYSTEMS, INC. ALL REASONABLE PRECAUTIONS SHALL BE TAKEN TO ENSURE COMPLIANCE WITH THESE CONDITIONS.

THE POSSESSION OF THIS DOCUMENT CONVEYS NO PROPERTY RIGHT OR LICENSE IN THE INFORMATION NOR IN ANY MATTER DISCLOSED THEREIN.

SLA-1

BETA SITE

STEREOLITHOGRAPHY

SYSTEM

USER'S MANUAL

TABLE OF CONTENTS

INTRODUCTION

DESCRIPTION AND GENERAL INFORMATION

CONTROLS AND INDICATORS

OPERATING INSTRUCTIONS

TROUBLESHOOTING

OPERATOR MAINTENANCE INSTRUCTIONS

3D Systems Incorporated
12847 Arroyo Street
Sylmar, California 91342
(818) 898-1533 • FAX (818) 361-5484

NOVEMBER 1987



systems, inc.

COMPANY CONFIDENTAL

Safety Precautions

WARNING
INVISIBLE LASER RADIATION

UV LASER RADIATION MAY DAMAGE EYE TISSUE. AVOID DIRECT EXPOSURE TO LASER BEAM. WEAR UV BLOCKING SAFETY GLASSES.

WARNING
HIGH VOLTAGE

HAZARDOUS VOLTAGES ARE ACCESSIBLE WITH MAINTENANCE PANELS REMOVED. FOLLOW SAFE WORK PRACTICES.

WARNING
CHEMICAL HAZARDS

UV CURABLE RESINS MAY CAUSE EYE AND SKIN BURNS. REPEATED OR PROLONGED SKIN CONTACT MAY CAUSE SENSITIZATION. VAPOR MAY BE HARMFUL.

1

systems, inc.

COMPANY CONFIDENTAL

Observe the following safety precautions when cleaning spilled resin or handling resin containers.

- Keep away from heat, sparks and flame. Protect from sunlight and fluorescent light. Closed containers may rupture/explode when exposed to extreme heat. Use National Fire Protection Association Class B extinguishers (carbon dioxide, dry chemical or foam).
- Use only with adequate ventilation. Avoid breathing vapors or spray mist.
- Wear safety glasses.
- Wear chemically resistant gloves and protective clothing. Wash thoroughly after handling and before eating, smoking or using toilet facilities.
- Wear an NIOSH/MSHA approved respirator or ensure adequate ventilation when sanding or cutting cured objects.

FIRST AID FOR CHEMICAL ACCIDENTS

Skin Contact. Wash thoroughly with soap and water. Remove contaminated clothing and shoes immediately. If skin is irritated, get medical attention. Wash clothing before reuse and discard contaminated shoes.

Eye Contact. Flush immediately with large amounts of water for 15 minutes and avoid sunlight, fluorescent light, or other ultraviolet light. Get medical attention.

Inhalation. Remove victim to fresh air. Give artificial respiration or cardio-pulminary resuscitation if required. If breathing is difficult, give oxygen. Get medical attention.

systems, inc.

COMPANY CONFIDENTAL

TABLE OF CONTENTS

Section/Paragraph	Title	Page
	INTRODUCTION	vi
I	DESCRIPTION AND OPERATION	1-1
1.1	Purpose	1-1
1.2	Description	1-1
1.2.1	Stereolithography Process	1-1
1.2.2	Stereolithography System	1-4
1.2.2.1	Electronic Cabinet Assembly	1-8
1.2.2.2	Optics Assembly	1-8
1.2.2.3	Chamber Assembly	1-9
1.2.3	Software	1-10
1.3	Performance Specification	1-12
II	CONTROLS AND INDICATORS	2-1
2.1	Introduction	2-1
2.2	Control Panel	2-1
III	OPERATING INSTRUCTIONS	3-1
3.1	Introduction	3-1
3.2	Materials and Equipment	3-1
3.3	CAD Design to SLA-1 Parts	3-2
3.3.1	How to Design CAD Parts for SLA-1 . .	3-2
3.3.2	Rules for Design of Parts	3-2
3.3.3	How to Design Support Files for SLA-1	3-3
3.3.4	Rules for Design of Supports	3-3
3.4	Slice Operation	3-3
3.4.1	How to Slice Files	3-3
3.4.2	Rules for Slicing	3-4
3.4.3	How to Run User Interface	3-4
3.5	SLA-1 Operation	3-7
3.5.1	Turn-On Procedure	3-7
3.5.2	How to Transfer Files form Slice Computer to Process Computer	3-7
3.5.3	How to Insert Critical Volumes	3-8
3.5.4	How to Merge Sliced Files	3-10
3.5.5	How to Operate SLA-1 to Build Parts .	3-10
3.5.5.1	Rules for Building SLA-1 Parts	3-10
3.5.5.2	How to Edit Part Control Files	3-11
3.5.5.3	How to Prepare Default Parameter Files .	3-13
3.5.5.4	Operator's Checklist Before Starting a Part	3-14
3.5.5.5	How to Run Supervisor to Make a Part .	3-16

TABLE OF CONTENTS - CONT

Section/Paragraph	Title	Page
3.5.5.6	Post Processing of SLA-1 Parts	3-17
3.5.5.7	Shutdown Procedures	3-20
3.5.5.8	How to Create and Use Working Curves .	3-20
IV	TROUBLESHOOTING	4-1
4.1	Introduction	4-1
4.2	Troubleshooting Procedures	4-1
V	OPERATOR MAINTENANCE INSTRUCTIONS . .	5-1
5.1	Introduction	5-1
5.2	Materials and Equipment	5-1
5.3	Resin Cleaning/Replenishment Procedures	5-2
5.3.1	Cleanup of Small Resin Spills	5-2
5.3.2	Cleanup of Large Resin Spills	5-3
5.3.3	Vat Replenishment	5-4
5.4	Optics Cleaning	5-4
5.4.1	Equipment Required	5-4
5.4.2	Care and Handling of Optics	5-7
5.4.3	Laser Brewster Windows	5-7
5.4.4	Laser Resinator Windows	5-7
5.4.5	90 Degree Beam-Turning Mirrors	5-7
5.4.6	Beam Expander Lenses	5-8
5.4.7	Galvanometer-Driven Dynamic Mirrors .	5-10
5.4.8	Laser Chamber Window	5-11
5.5	Replacement Procedures	5-13
5.5.1	Air Filter Replacement	5-13
5.5.2	Chamber Lamp	5-13
5.5.3	Control Panel Lamps	5-15
	GLOSSARY	G-1



systems, inc.

COMPANY CONFIDENTAL

LIST OF FIGURES

Figure	Title	Page
1-1	Key Steps in the Stereolithography Process	1-2
1-2	Major Components of the Stereolithography System	1-5
1-3	Block Diagram of the Stereolithography System	1-7
1-4	Software Diagram of the Stereolithography System	1-11
2-1	Control Panel Switches and Indicators	2-2
3-1	Sample Part Log	3-12
3-1	Sample Working Curve	3-22
5-1	Recommended Optics Cleaning Technique	5-5
5-2	Air Filter Replacement	5-14

LIST OF TABLES

Figure	Title	Page
1-1	Beta SLA-1 Performance Specifications	1-12
4-1	Troubleshooting Procedures	4-2
5-1	Maintenance Materials and Equipment .	5-1



systems, inc.

COMPANY CONFIDENTAL

INTRODUCTION

SCOPE

This manual contains operation and maintenance procedures and information for the SLA-1 Beta Site Stereolithography System. The maintenance procedures are limited in scope to routine maintenance to be performed by the SLA-1 operator, such as filter replacement. Service and repairs not covered in this manual shall be performed by qualified technicians in accordance with the SLA-1 Service Manual.

ORGANIZATION

This manual is divided into five sections, as follows:

SECTION I. DESCRIPTION AND GENERAL INFORMATION - Physical and functional descriptions of the SLA-1 and a list of performance specifications.

SECTION II. CONTROLS AND INDICATORS - Description of each operator control and indicator.

SECTION III. OPERATING INSTRUCTIONS - Operating procedures and information.

SECTION IV. TROUBLESHOOTING - Diagnostic and correction procedures.

SECTION V. OPERATOR MAINTENANCE INSTRUCTIONS - Procedures for routine maintenance.

OTHER FEATURES OF INTEREST

Title Page Index. On the right side of the title page you will find a section index. The black markers printed to the right of the text are keyed to similar markers on the first page of each referenced section. To locate a section, flip through the manual until you find a marker printed in the same location as the title page marker.



systems, inc.

COMPANY CONFIDENTAL

Warning Pages. Following the title page you will find a summary of critical warnings. You may be seriously injured if these warnings are not explicitly followed. Carefully read all warnings and first aid instructions prior to operation or maintenance of the SLA-1.

REPORTING ERRORS AND RECOMMENDING IMPROVEMENTS

This manual was prepared for use by Beta Site users and maintenance personnel. It will be revised to incorporate recommended content and format changes. If you find any mistakes or if you have any recommendations for improving the procedures, please let us know by sending a copy of the marked-up page(s) to:

Attn: Chick Lewis
3D Systems Inc.
12847 Arroyo St.
Sylmar, California 91342
(818) 898-1533 • FAX (818) 361-5484



systems, inc.

COMPANY CONFIDENTAL

SECTION I

DESCRIPTION AND GENERAL INFORMATION

1.1 PURPOSE

The SLA-1 Stereolithography System produces three-dimensional parts directly from a CAD system. The formed objects, up to nine inches in any dimension, are made of photocurable plastic. They can be used in a wide variety of applications, including:

- Industrial Engineering
- Design Engineering
- Architectural Design
- Medical
- Scientific

1.2 DESCRIPTION

1.2.1 Stereolithography Process. Stereolithography is a three-dimensional printing process which uses a moving laser beam to build parts by solidifying successive layers of liquid plastic. This method enables a designer to create a design on a CAD system and build an accurate plastic model in a few hours. The stereolithographic process is composed of the following eight steps, as shown in Figure 1-1.

- Design solid model
- Prepare model for stereolithography
- Section model into triangles and reduce data for transmission
- Transmit data file to SLA-1 slice computer
- Slice triangle files horizontally
- Calculate vectors and add hatch and fill
- Form object
- Post processing

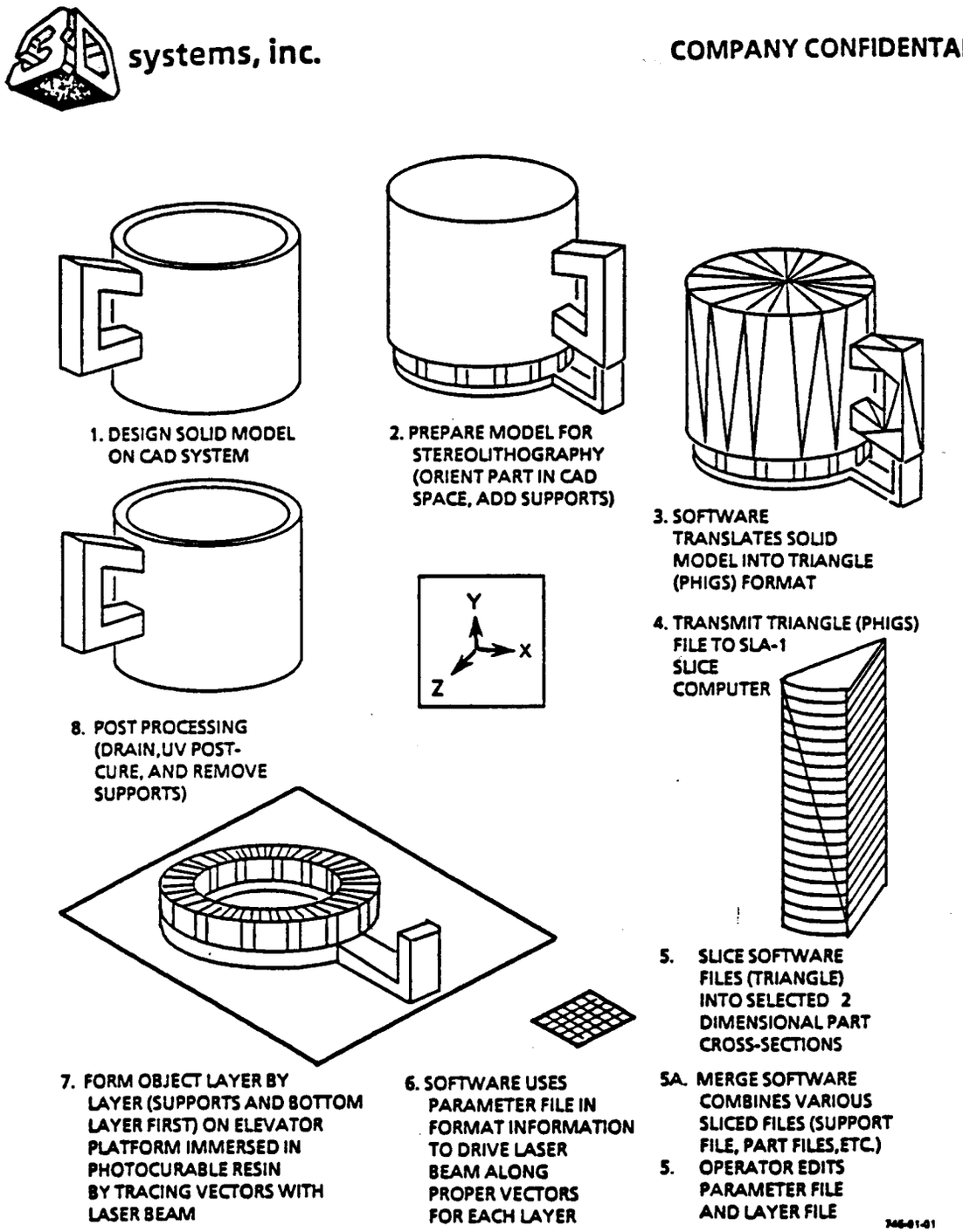


Figure 1-1. Key Steps in the Stereolithography Process



systems, inc.

COMPANY CONFIDENTAL

1. The solid model is designed in the normal way on the CAD system, without specific reference to the stereolithographic process. A copy of the model is made for stereolithographic processing.

2. Model preparation for stereolithography involves selecting the optimum orientation, adding supports, and selecting SLA-1 operating parameters. The optimum orientation will (1) enable the object to drain, (2) have the least number of unsupported surfaces, (3) optimize important surfaces, and (4) enable the object to fit in the resin vat. Supports must be added to secure unattached sections and for other purposes; a CAD library of supports can be prepared for this purpose. SLA-1 operating parameters include selection of the model scale and layer (slice) thickness.

3. The surface of the solid model is then divided into triangles. A triangle is the least complex polygon for vector calculations. The Beta SLA-1 capability is approaching 200,000 triangles, with further improvements planned for the production SLA-1. The more triangles formed, the better the surface resolution and hence the more accurate the formed object with repect to the CAD design.

4. Data points representing the triangle coordinates are then transmitted to the SLA-1 via Ethernet communication. The SLA-1 software slices the triangular sections horizontally (X-Y plane) at the selected layer thickness.

5. The SLA-1 next calculates the section boundary, hatch, and horizontal surface (skin) vectors. Hatch vectors consist of cross-hatching between the boundary vectors. Several styles are available. Skin vectors, which are traced at high speed and with a large overlap, form the outside horizontal surfaces of the object. Interior horizontal areas, those within top and bottom skins, are not filled in other than by cross-hatch vectors.



systems, inc.

COMPANY CONFIDENTAL

6. The SLA-1 forms the object one horizontal layer at a time by moving the ultraviolet beam of a helium-cadmium laser across the surface of a photocurable resin and solidifying the liquid where it strikes. Absorption in the resin prevents the laser light from penetrating deeply and allows a thin layer to be formed. Each layer is comprised of vectors which are drawn in the following order: border, hatch, and surface.

7. The first layer that is drawn adheres to a horizontal platform located just below the liquid surface. This platform is attached to an elevator which then lowers it vertically under computer control. After drawing a layer, the platform dips several millimeters into the liquid to coat the previous cured layer with fresh liquid, then rises up a smaller distance leaving a thin film of liquid from which the second layer will be formed. After a pause to allow the liquid surface to flatten out, the next layer is drawn. Since the resin has adhesive properties, the second layer becomes firmly attached to the first. This process is repeated until all the layers have been drawn and the entire three-dimensional object is formed. Normally, the bottom 0.25 inch or so of the object is a support structure on which the desired part is built. Resin that has not been exposed to light remains in the vat to be used for the next part. There is very little waste of material.

8. Post processing involves heating the formed object to remove excess resin, ultraviolet or heat curing to complete polymerization, and removing supports. Additional processing, including sanding and assembly into working models, may also be performed.

1.2.2 Stereolithography System. The SLA-1 is a self-contained system that interfaces directly with the user's CAD system. The SLA-1, as shown in Figure 1-2, consists of four major component groups: the slice computer terminal, the electronic cabinet assembly, the optics assembly, and the chamber assembly. A block diagram of the SLA-1 is shown in Figure 1-3.

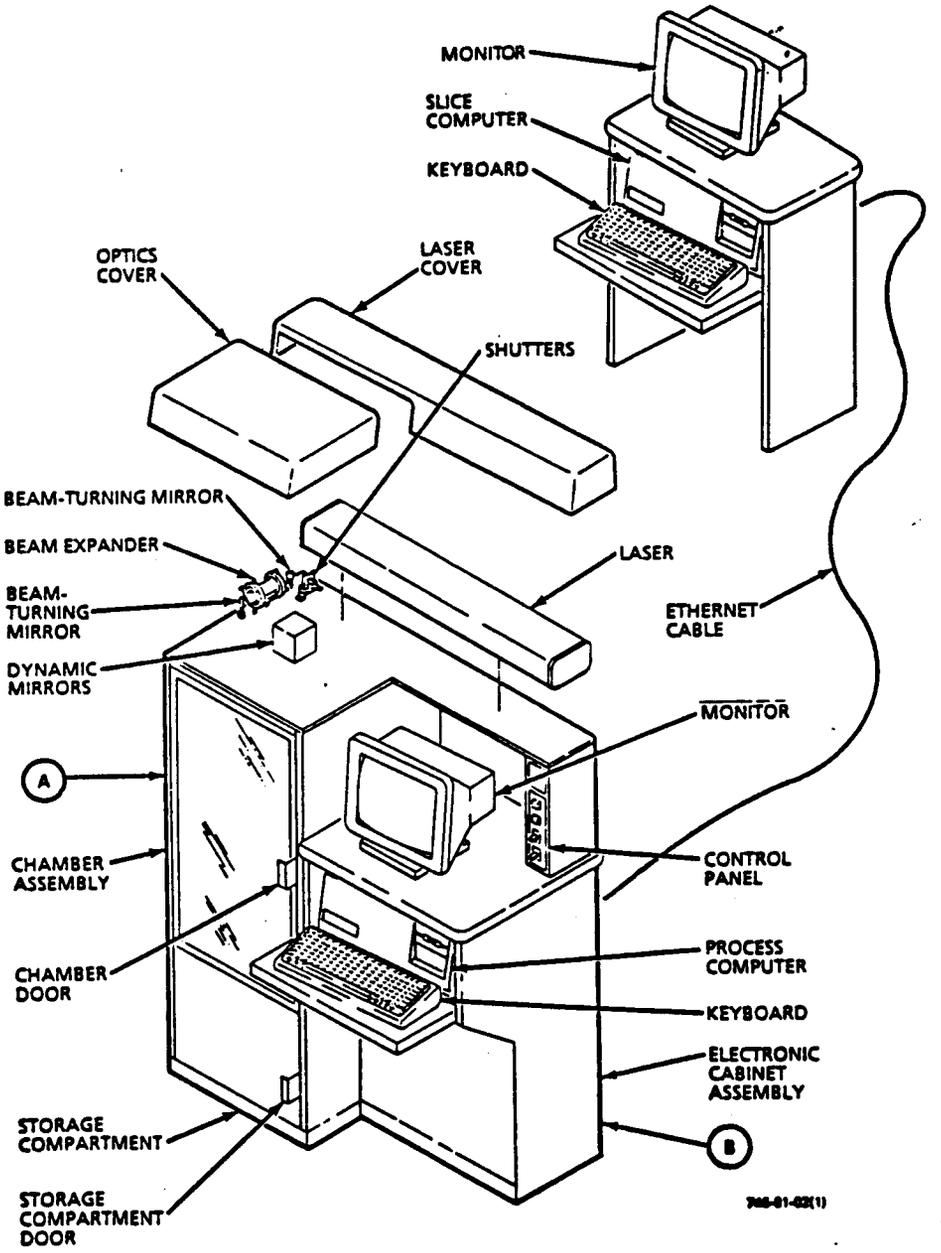


Figure 1-2. Major Components of the Stereolithography System (Sheet 1 of 2)

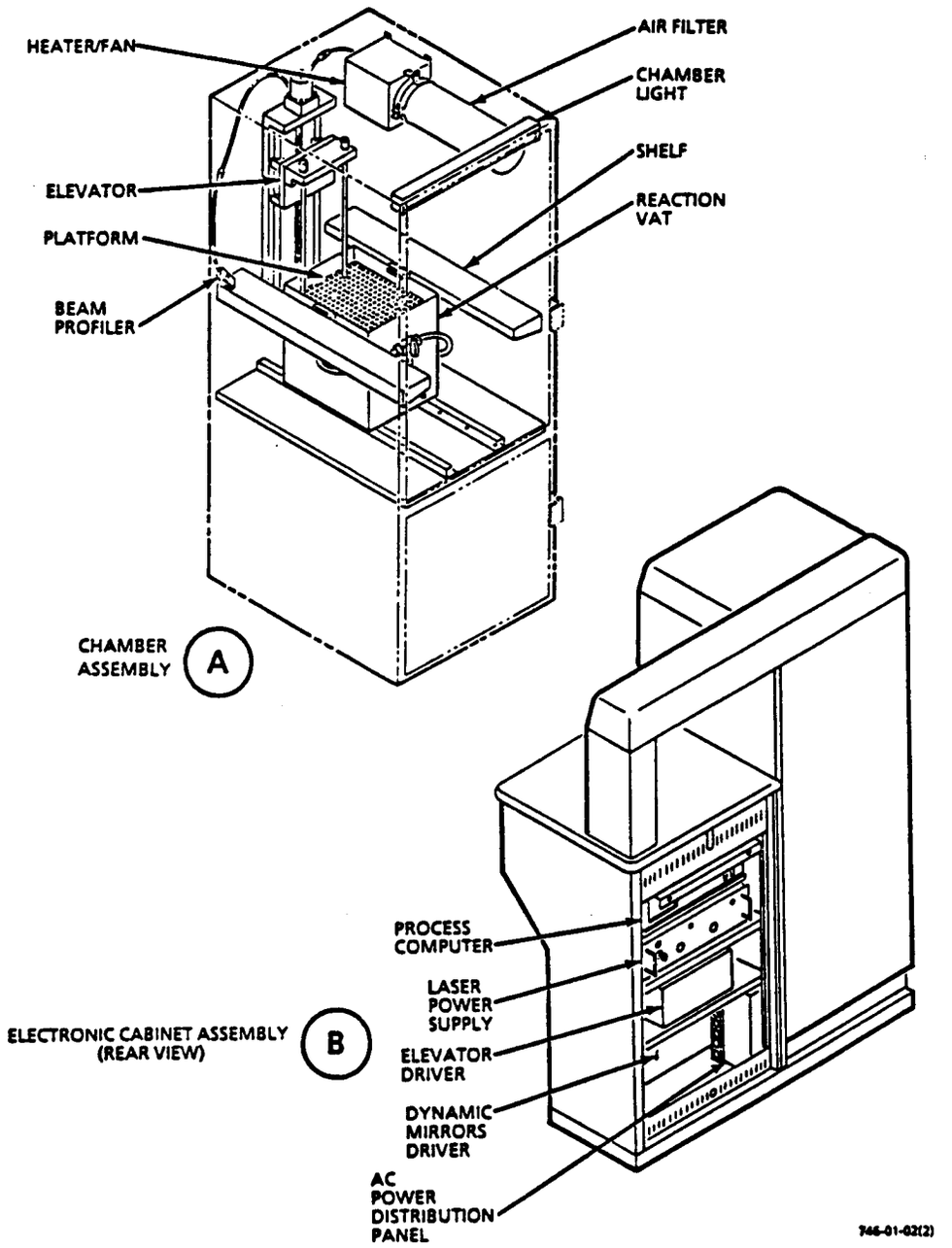


Figure 1-2. Major Components of the Stereolithography System (Sheet 2 of 2)

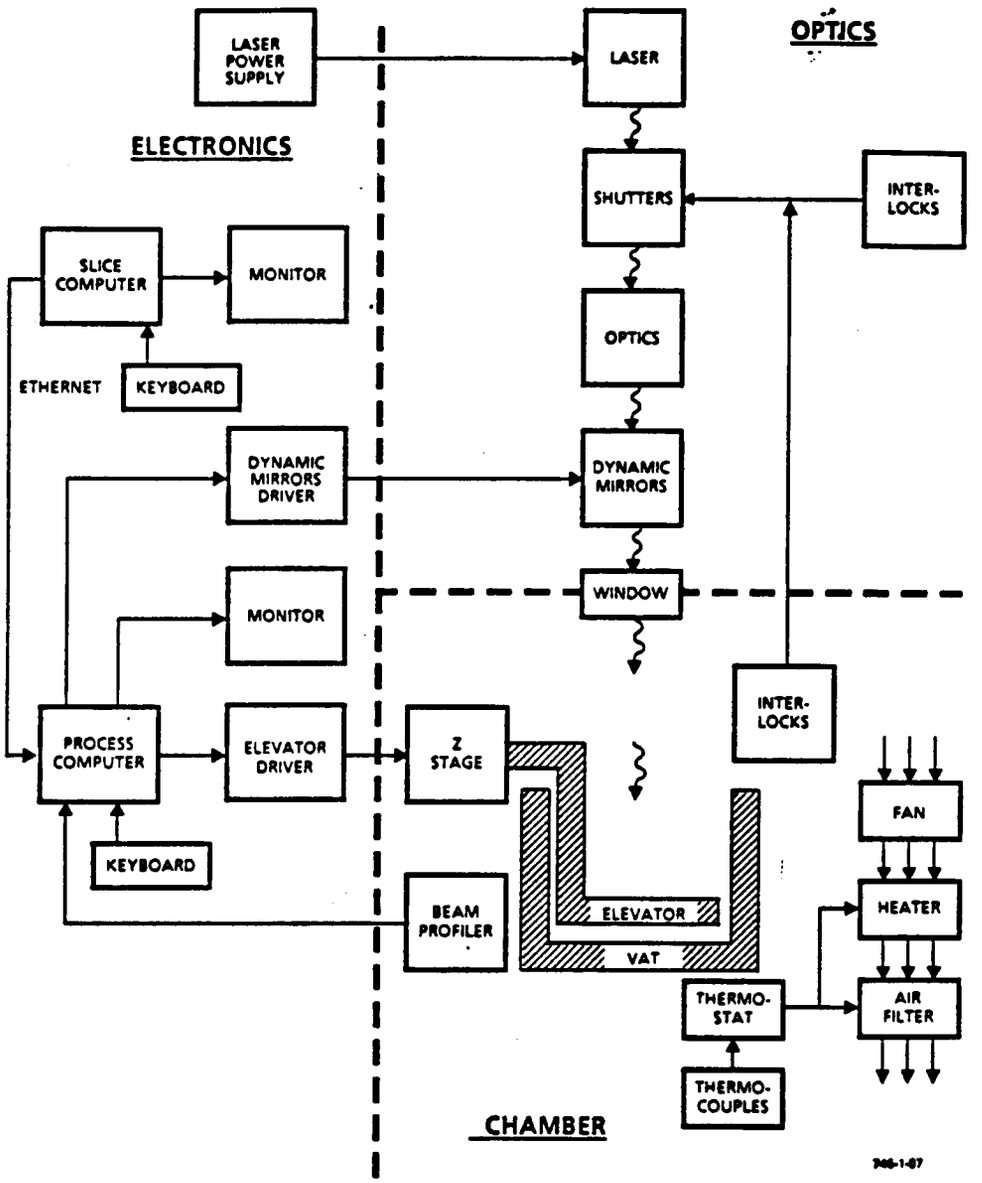


Figure 1-3. Block Diagram of the Stereolithography System

1.2.2.1 Electronic Cabinet Assembly. The electronic cabinet assembly includes the process computer (disc drive), keyboard, monitor, power supplies, ac power distribution panel, and control panel. The computer assembly includes plug-in circuit boards for control of the terminal, high-speed scanner mirrors, and vertical (Z-stage) elevator. Power supplies for the laser, dynamic mirrors, and elevator motor are mounted in the lower portion of the cabinet.

The control panel includes a power on switch/indicator, a chamber light switch/indicator, a laser on indicator, and a shutter open indicator.

Operation and maintenance parameters, including fault diagnostics and laser performance information, are displayed on the monitor. Operation is controlled by keyboard entries. Work surfaces around the key-board and disc drive are covered with formica for easy cleaning and long wear.

1.2.2.2 Optics Assembly. The helium cadmium (HeCd) laser and optical components are mounted on top of the electronic cabinet and chamber assembly. The laser and optics plate may be accessed for service by removing separate covers. For safety reasons, a special tool is required to unlock the cover fasteners and interlock switches are activated when the covers are removed. The interlocks activate a solenoid-controlled shutter to block the laser beam when either cover is removed.

An optics cleaning kit and interlock shorting tool are located under the optics cover. The cleaning kit includes cotton swabs, special cleaning tissues, and materials for cleaning the beam-turning mirrors and beam expander lenses. The interlock shorting tool is used to defeat the interlocks during servicing. This is necessary to enable optics alignment and servicing procedures which require operation of the laser with the optics and laser covers removed.

The shutter assembly, two 90° beam-turning mirrors, beam expander, scanning mirror assembly, and optical window are mounted on the optics plate. The rotary solenoid-actuated shutters are installed at the laser output and rotate to block the beam when a safety interlock is opened. The 90° beam-turning mirrors reflect the laser beam to the next optical

component. The beam expander enlarges and focuses the laser beam on the liquid surface. The high speed scanning mirrors direct the laser beam to trace vectors on the resin surface. A quartz window between the optics enclosure and reaction chamber allows the laser beam to pass into the reaction chamber, but otherwise isolates the two regions.

1.2.2.3 Chamber Assembly. The chamber assembly contains an environmentally-controlled chamber, which houses a platform, reaction vat, elevator, and beam profiler.

The chamber in which the object is formed is designed for operator safety and to ensure uniform operating conditions. The chamber may be heated to approximately 40°C (104°F) and the air is circulated and filtered. An overhead light illuminates the reaction vat and work surfaces. An interlock on the glass access door activates a shutter to block the laser beam when opened.

The reaction vat is designed to minimize handling of the resin. It is installed in the chamber on guides which align it with the elevator and platform.

The part is formed on a platform attached to the vertical axis elevator, or Z-stage. The platform is immersed in the resin vat and is adjusted incrementally downward while the object is being formed. To remove the formed part, it is raised to a position above the vat. The platform is then disconnected from the elevator and removed from the chamber for post processing. Handling trays are provided to catch dripping resin.

The beam profiler is mounted to one side of the reaction vat at the focal length of the laser. The scanning mirror is periodically commanded to direct the laser beam onto the beam profiler, which measures the beam intensity profile. The data may be displayed on the terminal either as a profile with intensity contour lines or as a single number representing the overall (integrated) beam intensity. This information is used to determine whether the mirrors should be cleaned and aligned, whether the laser should be serviced, and what parameter values will yield vectors of the desired thickness and width.

1.2.3 Software. A software diagram of the SLA-1 is shown in Figure 1-4. There are three computers needed to control the stereolithographic apparatus, a CAD system, a slice computer, and a process computer. Any CAD system can be used to design a part in three-dimensional space. This is identified as the object file. In order to generate the part, supports must be added to prevent distortion. This is accomplished by adding the necessary supports to the CAD part design and creating a CAD support file. The resultant two or more CAD generated files are then physically inserted into the slice computer through Ethernet.

The stereolithography apparatus builds the part one layer at a time starting with the bottom layer. The slice computer breaks down the CAD part into individual horizontal slices. The slice computer also calculates where hatch vectors will be created. This is done to achieve maximum strength as each layer is constructed. The slice computer at the Beta Sites is a separate computer with its own keyboard and monitor. It is anticipated that in production models, the slice computer will be in the SLA-1 electronic cabinet and will share a common keyboard and monitor with the process computer. The operator can vary the thickness of each slice and change other parameters of each slice with the User Interface program. The slice computer uses the Xenix machine language and is connected to the SLA-1 process computer by an Ethernet data bus.

The sliced files are then transferred to the process computer through Ethernet. The process computer merges the sliced object and support files into a layer control file and a vector file. The operator then inserts the necessary controls needed to drive the stereolithography apparatus in the layer and parameter file. (The vector file is not usually edited.) The operator can strengthen a particular volume of the part by inserting rivets. This is accomplished by inserting the necessary parameters to the critical volume file prior to merging of the sliced files. The merge program integrates the object, support, and critical volume files and inserts the resultant data in the layer control file. The operator can edit the layer control file and change the default parameter file. The default parameter file contains the controls needed to operate the stereolithography apparatus to build the part. The process computer uses the MSDOS machine language and is directly connected to he stereolithography apparatus.

systems, inc.

COMPANY CONFIDENTAL

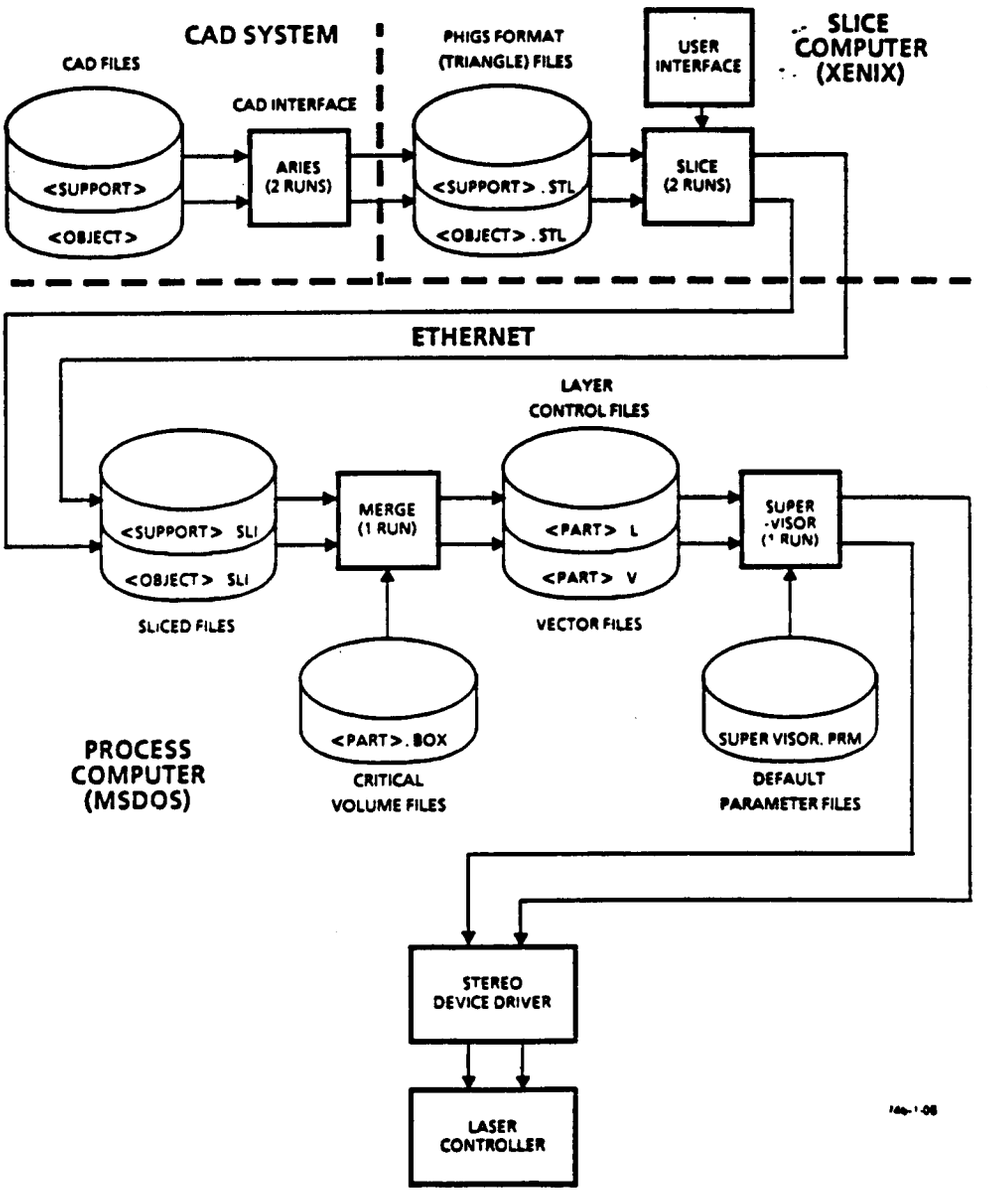


Figure 1-4. Software Diagram of Stereolithography System



systems, inc.

COMPANY CONFIDENTAL

1.3 PERFORMANCE SPECIFICATIONS

Beta SLA-1 performance specifications are listed in Table 1-1 for quick reference.

Table 1-1. Beta SLA-1 Performance Specifications

Characteristic	Specification
Input Power	
Voltage and Frequency	115 Vac, 60 Hz
Current Drain	15 amperes
Functional Characteristics	
Laser Source	
Type	Helium Cadmium
Nominal Power	10 mW at 325 nm
Warmup Time (to 90 percent power)	15 minutes
Expected Tube Lifetime	4000 hours (with 1000 hour service intervals)
Dielectric Beam-Turning Mirrors	
Nominal Reflectivity	>99 percent at 325 nm
Incident Angle Range	0 to 45 degrees
Beam Expander	
Expansion Coefficient	4
Adjustable Axes	Y,Z (pitch and yaw)
Scanning Mirror	
Speed	
Minimum	0.001 inch/sec (0.025 mm/sec)
Maximum	20 inches/sec (51 cm/sec)
Drawing Area	
Width (X-axis)	9 inches (23 cm)
Length (Y-axis)	9 inches (23 cm)
Elevator (Z-stage)	
Maximum Travel	14 inches (36 cm)
Minimum Step Size	0.001 inch (0.025 mm)
Load Capacity	110 lbs (___ kg)
CAD Interface	Ethernet
Resin Characteristics	
Appearance and Odor	Clear amber liquid, mild odor
Vapor Density	Heavier than air
Specific Gravity (H_2O=1)	1.18
Flash Point	>200°F (93°C)
Extinguishing Media	National Fire Protection Association Class B Extinguishers (carbon dioxide, dry chemical or foam)
Storage Temperature	+60°F to +80°F (+15°C to +26°C)



systems, inc.

COMPANY CONFIDENTAL

Table 1-1. Beta SLA-1 Performance Specifications - CONT

Characteristic	Specification
Physical Characteristics	
SLA-1	
Dimensions (maximum)	
Height	64 inches (163 cm)
Width	49 inches (124 cm)
Depth	27.5 inches (70 cm)
Shipping Weight	600 lbs (___ kg)
Resin Vat	
Dimensions	
Height	10 inches (25 cm)
Width	9 inches (23 cm)
Depth	9 inches (23 cm)
Operating Conditions	
Temperature	50°F to 104°F (10°C to 40°C)
Relative Humidity	0 to 90 percent



systems, inc.

COMPANY CONFIDENTAL

SECTION II

CONTROLS AND INDICATORS

2.1 INTRODUCTION

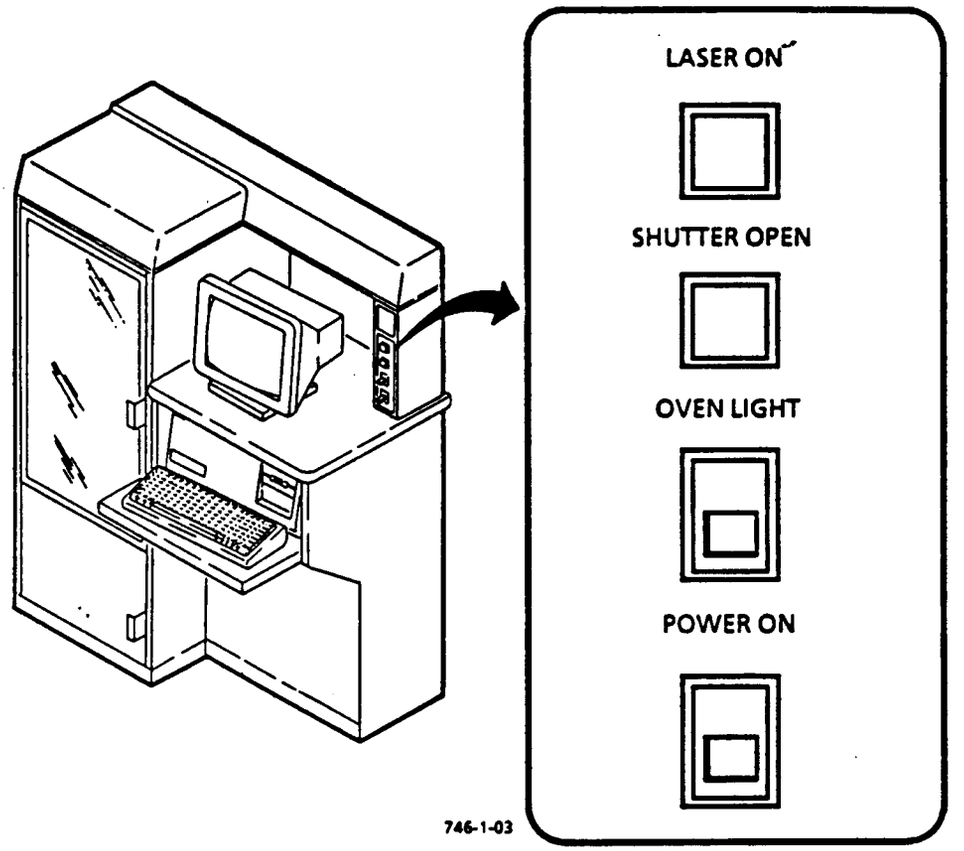
This section describes the function of each SLA-1 operating control and indicator.

2.2 CONTROL PANEL

Control panel switches and status indicators are illustrated and described in Figure 2-1.

systems, inc.

COMPANY CONFIDENTAL



Nomenclature	Function
LASER ON indicator	Blue lamp indicates that the laser is operating when illuminated.
SHUTTER OPEN indicator	Green lamp indicates that the laser shutter is open when illuminated.
OVEN LIGHT toggle switch/indicator	Turns the overhead reaction chamber light on and off.
POWER ON toggle switch/indicator	Controls primary power application to the laser, power supplies, reaction chamber, filter/heater, and process computer.

Figure 2-1. Control Panel Switches and Indicators



systems, inc.

COMPANY CONFIDENTAL

SECTION III

OPERATING INSTRUCTIONS

3.1 INTRODUCTION

This section contains complete operating instructions for design and construction of model parts. Included are instructions for design of parts and supports on the CAD system, slicing of parts on the slice computer, and operation and control of the SLA-1 system to build parts. Also included are instructions for file transfer, insertion of critical volumes, merging of sliced files, editing of part control files, preparing default parameter files, running supervisor to build parts, post processing of parts, and use of working curves.

3.2 MATERIALS AND EQUIPMENT

Materials and equipment that would enhance the operation of the SLA-1 system are listed in Table 3-1. Equivalent items may be used.

Table 3-1. Operation Materials and Equipment

Nomenclature	Model/Part No.	Use
Printer, Computer		Print out computer data permanent record file.
Microscope		Make working curve measurements.
Oven, Heat		Post cure green parts.
Oven, UV Flood		Post cure green parts.



systems, inc.

COMPANY CONFIDENTAL

3.3 CAD DESIGN FOR SLA-1 PARTS

3.3.1 How to Design CAD Parts for SLA-1. A part must initially be designed on a CAD system before it can be built on the SLA-1 system. This manual assumes that the operator knows how to design parts using a CAD system. In order to make the CAD design compatible with the SLA-1 system, the operator usually prepares two or more files on the CAD system, object files and support files. The object file is simply the CAD part. The support file is needed to add supporting structures to enable the part to keep its shape during construction on the SLA-1 system.

3.3.2 Rules for Design of Parts. In order to prepare the CAD design for the SLA-1 system, the operator should modify the CAD object file as follows:

a. Wall thicknesses should ideally be 0.020 to 0.150 inch.

b. Rotate CAD parts into an orientation which will:

   1. Minimize trapped volumes as the part builds.

   2. Take advantage of nice up-facing skin surfaces.

   3. Minimize visibility of down-facing skin surfaces.

   4. Make support design easy and optimum.

   5. Make parts stable and strong as they are created.

c. Design parts so that horizontal gaps and holes are larger than desired by one laser linewidth.

d. All solid parts must completely enclose a volume. Single surfaces will confuse the crosshatch algorithms.



3.3.3 How to Design Support Files for SLA-1. Supporting structures consist of bases, posts, and webs which are needed to properly support the part and prevent the part from being distorted during construction on the SLA-1 system. The supports should be designed on the CAD system in a separate support file.

3.3.4 Rules for Design of Supports. The operator should create the CAD support file as follows:

a. Design a structure at the bottom of the CAD part which will rest on the elevator panel. This platform must have several horizontal legs which are at least 0.65 inch long (more than twice the diameter of the 1/4 inch holes in the platform).

b. Design supports to intersect every outside corner of a part, as these are major stress areas.

c. Arrange supports so that every unsupported down-facing border falls on a previously created support.

d. Space supports at a minimum distance apart for best stress resistance.

e. Design supports to have at least two layers of vertical overlap into the part for a strong attachment.

3.4 SLICE OPERATION

3.4.1 How to Slice Files. The slice computer (Wyse PC 386) automatically breaks down the object and support files into individual slices under user control. The user must choose the thickness of each slice and determine the shape and method of cross-hatching.

3.4.2 Rules for Slicing. The operator should slice the CAD object and support files as follows:

a. Up-facing skin surfaces should only be in one dimension (X or Y) and should have a 0.002 inch offset (high overlap). Exposure must be low.



b. Cross hatch should usually be as nearly perpendicular to part border as possible. Cross hatches parallel to part borders increase production time and may increase stress.

c. Slice support files without creating skin.

3.4.3 How to Run User Interface. This procedure shows how to operate the slice computer to use the User Interface program to insert slice parameters and run the Slice program. This procedure assumes that the CAD file has been installed in the slice computer. An asterisk (*) in front of a step indicates that this is an optional step that only needs to be performed if a common keyboard is being used to operate the slice computer and process computer.

a. Press ENTER - MAIN MENU is displayed.

*b. Select Data Transfer (Slice) and press ENTER - Data Transfer menu is displayed.

*c. Select TELNET, Terminal Utility and press ENTER.

d. In response to $ prompt, type UI (user interface) and press ENTER - SLICE USER INTERFACE Menu is displayed.

e. Select option 1 (DATABASE File Name).

f. In response to Enter Data File Name: prompt, type data file name followed by .stl (example - test.stl) and press ENTER.

g. In response to Type File Binary or ASCII (B,A): prompt, type b (binary) or a (ASCII) as applicable and press ENTER.

h. Select option 2 (Scale).

i. In response to Enter Scale Value: prompt, type (scale value per CAD dimension unit) and press ENTER. (If 1000 is selected, 1000.000 is inserted in the Value column, which is 1/1000 of one CAD dimension unit.) (Example - for a part CAD designed in inches, a scale of 1000 makes each slice scale unit 1 mil.)

j. Select option 3 (Z Spacing).

k. In response to Enter Fixed or Variable Spacing (F, V, or Q) value: prompt, type F (fixed) and press ENTER. Then type thickness in slice scale units (from option 2) (example - 20) and press ENTER. (To select variable thicknesses, refer to software manual.)

l. Select option 4 (X hatch spacing).

m. In response to Enter Hatch Spacing (hx) value: prompt, type X hatch spacing in slice scale units (example - 200 (5 hatches/inch)) and press ENTER.

NOTE

Do not use option 5 (Y hatch spacing) if option 6 (60/120 degree hatch spacing) is to be used.

n. Select option 5 (Y hatch spacing).

o. In response to Enter Hatch Spacing (hy) value: prompt, type Y hatch spacing in slice scale units (example - 200) and press ENTER.

p. Select option 6 (60/120 degree hatch spacing).

q. In response to Enter Hatch Spacing (60/120) value: prompt, type 60/120 hatch spacing in slice scale units (example - 20) and press ENTER.

r. Select option 7 (X Skin fill for near flat surfaces).

s. In response to Enter Skin fill for near flat surfaces (hfx) value: prompt, type X skin fill offset in slice scale units (example - 2) and press ENTER.

t. SELECT option 8 (Y Skin fill for near flat surfaces).

NOTE

If X skin fill is used, Y should not be used, and vice versa.

u. In response to Enter Skin fill for near flat surfaces (hfy) value: prompt, type Y skin fill in mils (example - 2) and press ENTER.

v. Select option 9 (minimum Surface Angle for scanned facets).

w. In response to Enter a Minimum Surface Angle prompt, type desired angle in degrees from vertical (example - 60) and press ENTER.

x. Select option 10 (Minimum Hatch Intersect Angle).

y. In response to Enter a Minimum Intersect Angle value: prompt, type intersect angle in degrees (example - 20) and press ENTER.

z. Select option 11 (Segment Output File Name).

aa. In response to Enter Segment File Name: prompt, type desired output file name followed by .sli (slice) (example - test.sli) and press ENTER.

ab. After all slice parameters have been selected, select s (Save) and press ENTER. (This will save the parameters for future use and reference.)

ac. In response to "Press (Enter) to Continue" prompt, press ENTER. Then select d (DoSlice) and press ENTER.

ad. In response to Slice Version to use (Default XY)? prompt, press ENTER. (The program now slices the files using the inserted slice parameters.)

ae. After slicing is completed, DATA TRANSFER MENU is displayed.

af. Press Q (Quit) and ENTER. (The sliced files are now ready to be transferred to the process computer.)

3.5 SLA-1 OPERATION

3.5.1 Turn-On Procedure.

a. Set POWER ON switch to on (up). Verify that POWER ON indicator illuminates.

b. Set OVEN LIGHT switch to on (up). Verify that OVEN LIGHT indicator illuminates and that overhead light in reaction chamber illuminates.

NOTE

The SHUTTER OPEN and LASER ON indicators will illuminate during operation. The SHUTTER OPEN indicator will illuminate when the laser shutter is open and the LASER ON indicator will illuminate when the laser is operating.

c. The MAIN MENU will be displayed on the monitor as the process computer boots up. Select "Power on sequence" and press ENTER.

d. The POWER SEQUENCE menu will be displayed. Sequentially press function keys 1, 2, and 3 to power up the laser, mirror, and elevator drivers, and to open the laser shutters.

e. Wait at least 15 minutes for the laser to stabilize in power before beginning to create a part. Other functions (file preparation, data transfer, etc.) can be performed during laser warm-up.

3.5.2 How to Transfer Files from Slice Computer to Process Computer. This procedure shows how to transfer the sliced object and support files from the slice computer to the process computer (Wyse PC 286) in the SLA-1.

a. Press ENTER - MAIN MENU is displayed.

b. Select option 1 (Data Transfer).

c. In response to (data transfer) prompt, type 2 (FTP) (file transfer program) and press ENTER.

d. In response to (ftp) prompt, type OPEN and press ENTER.

e. In response to (to) prompt, type address of slice computer and press ENTER.

f. In response to Remote user prompt, type the name of your directory and press ENTER.

g. In response to Password prompt, type your password and press ENTER.

h. In response to (ftp) prompt, type GET and press ENTER.

i. In response to (remote-file) prompt, type name of desired file usually followed by .sli (example - test.sli) and press ENTER.

j. In response to (local-file test.sli is default) prompt, press ENTER (unless you want to change name). (The FTP routine will now transfer the file to the process computer. It will prompt when the transfer is complete.)

k. To exit from FTP, in response to ftp) prompt, type BYE and press ENTER. (The sliced files have now been transferred to the SLA-1 process computer.)

l. MAIN MENU is displayed after transfer is completed.

3.5.3 How to Insert Critical Volumes. This procedure shows how to set up critical volumes. These critical volumes can be used to insert rivets, which are multiple passes of the laser beam over crosshatch vectors to increase strength, or for other special treatment (This procedure can be skipped if the CAD part contains no critical volumes.)

a. On CAD computer, call up CAD display of the part.



b. Identify X, Y, and Z coordinates in CAD space for the four bottom corners of the rectangular solid (critical volume).

c. On process computer, select option 5 (Edit System Files) and press ENTER.

d. Select option to create a new file - Turbo Basic is displayed.

e. Use arrow keys to select Write to and press ENTER.

f. In response to New Name prompt, enter name of critical volume followed by .box (example - test.box) and press ENTER.

g. Use arrow keys to select Edit and press ENTER.

h. In response to C: Test .Box enter: prompt, insert the following:

<type>, <base>, <height>, <x1>, <y1>, <x2>, <y2>, <x3>, <y3>, <x4>, <y4> (Be sure to insert commas between each item. Proper syntax is critical.)

where:

<type> is "XV" to rivet cross-hatches in the enclosed area or "XI" to ignore cross-hatches

<base> is the base of the box relative to the slice scale
<height> is the height of the box
<x1, y1> is the first coordinate of the box
<x2, y2> is the second coordiante
<x3, y3> is the third coordinate
<x4, y4> is the fourth coordinate

i. Press Esc (escape) key.

j. Use arrow key to select File and press ENTER.

k. Use arrow key to select Save and press ENTER.

l. Use arrow key to select Quit and press ENTER. (The new <part> .box file has now been created to identify a critical volume.)

3.3.4 How to Merge Sliced Files. This procedure shows how to combine (merge) the object and support files into vector and layer control files.

a. Press ENTER - MAIN MENU is displayed.

b. Select option 2 (MERGE) and press ENTER.

c. In response to Slice File Names prompt, type names of files to be merged (include .SLI as part of the names) and press ENTER. (Be sure to include critical volume file, if applicable.)

d. In response to Output File Name prompt, type desired name of output file and press ENTER. (No ".XXX" ending is necessary.)

e. Press ENTER and wait for process computer to merge files (one slice at a time). (The program will prompt when it has completed the merge.)

3.5.5 How to Operate SLA-1 to Build Parts. These procedures show how to use the process computer to actually build a part in the reaction vat. Included are procedures for setting up the reaction vat, modifying the merged vector and control files, preparing the default parameter files, and running the part making (supervisor) program.

3.5.5.1. Rules for Building SLA-1 Parts. In order to prepare the SLA-1 to build a part, the operator must perform the operator's checklist, edit the layer control (.L) file (SUPER .PRM), prepare the default parameter files, and run the supervisor programs as follows:

a. Set speed of first support layer three times slower than normal layer drawing speed. This overcures first layer to enable it to adhere firmly to elevator platform.

b. Set dip speeds low to avoid undue stresses to part in progress.

c. Use longer dip delays for:

1. Fragile layers.

2. Lowest layers of supports near platform.

3. After layers with large area skins.

4. In areas with large trapped volumes of resin.

5. For shallow dip depths (thin layer thicknesses).

d. Use single passes and choose exposure speeds from working curves which provide a 0.006 to 0.008 inch overcure into previously created layers.

e. Record important parameters and comments in a Part Log (example in figure 3-1). (The user is encouraged to create his own custom Part Log to his special requirements.) If a printer is available, print out important parameters for a permanent record.

3.5.5.2 How to Prepare Default Parameter Files. This procedure shows how to prepare the default parameter (.PRM) files to control the part building access:

a. Press ENTER - MAIN MENU is displayed.

b. Select option 5 (Edit System Files) and press ENTER.

c. In response to Load File Name prompt, enter file names (insert SUPER.PRM only) and press ENTER.

d. Move arrow to Edit block and press ENTER. Values can now be inserted in default parameter (SUPER.PRM) files. (Refer to software manual for definitions of codes.)



systems, inc.

COMPANY CONFIDENTAL

SAMPLE PART LOG

Part No.	Date	Matl.	Temp.	Pgm Name	Relat Speed	Layer Thick	Laser Power	Expnd Setng	Line Width	Comments

Figure 3-1. Sample Part Log

e. To quit editing files:

1. Press Esc key.

2. Use arrow key to select File and press ENTER.

3. Use arrow key to select Save and press ENTER.

4. Press Q key - MAIN MENU is displayed.

3.5.5.3 How to Edit Layer Control Files. This procedure shows how to modify the layer (.L) files to fine tune the part building process:

a. Press ENTER - MAIN MENU is displayed.

b. Select option 5 (Edit System Files) and press ENTER.

c. In response to Load File Name prompt, enter file names (insert .L files only; .V files are not usually edited) and press ENTER.

d. Move arrow to Edit block and press ENTER. Values can now be changed on layer (.L) files. (Refer to software manual for definition of codes.)

e. To quit editing files:

1. Press Esc key.

2. Use arrow key to select File and press ENTER.

3. Use arrow key to select Save and press ENTER.

4. Press Q key - MAIN MENU is displayed.



systems, inc.

COMPANY CONFIDENTAL

3.5.5.4 Operator's Checklist Before Starting a Part. This procedure shows how to set up the reaction vat, laser, and elevator, and how to monitor the part during construction.

a. Is correct resin in the vat? If not, insert proper resin in vat:

1. Open chamber door.
2. Drain out resin.
3. Clean vat thoroughtly.
4. Fill vat with correct resin.
5. Close chamber door.
6. Wait 1/2 hour for resin level to stabilize.

b. Is resin level adequate? If not, add resin in vat:

1. Open vat door.
2. Add resin as required.
3. Close vat door.
4. Wait 1/2 hour for resin level to stabilize.

c. Is elevator level correct? If not, set level as follows:

1. On process computer, press ENTER - main menu is displayed.
2. Select Utilities menu and press ENTER.
3. Select Z-Stage Mover and press ENTER.
4. To move elevator up, press ↑ (up) arrow and press any key to stop movement.

5. To move elevator down, press ↓ (down) arrow and press any key to stop movement.

CAUTION

Do not allow elevator to hit top bracket or side of vat. Mechanical damage can result.

6. Toggle up ↑ arrow until liquid resin level is just below holes in the elevator platform.

7. Toggle ↓ arrow until liquid resin is level with holes (dimples in surface visible).

d. (Optional) Is laser power properly adjusted? Check laser power as follows:

1. On process computer, press ENTER - main menu is displayed.

2. Select Utilities menu and press ENTER.

3. Select Beam/Profiler option and press ENTER.

4. The prompt Compute the Beam Profile and Measure its Intensity will be displayed while the laser is being focused on the beam profiler in the chamber.

5. The prompt Display Power will be displayed when the laser beam power has been computed and the laser is returned to its normal position.

e. Are all necessary files in the active directory? Check directory for correctly named vector, layer, and default parameter files as follows:

1. Type DIR, space, part name, and .*, and press ENTER to display vector (.V) and layer (.L) files.

2. Type DIR, space, *.prm, and press ENTER to display default parameter (.PRM) files.

f. Are all the values inserted in the vector, layer, and default parameter files correct and with proper syntax? Refer to software manual for definitions of codes and syntax.

g. Is the elevator driving subroutine initialized? Initialize elevator drive subroutine as follows:

1. On process computer, press ENTER - main menu is displayed.

2. Select Utilities menu and press ENTER.

3. Select Power On Sequence and press ENTER.

4. Function keys 1, 2, and 3 can be used to power the laser, driving electronics, and open the shutters.

3.5.5.5 How to Run Supervisor to Make a Part. Now that all of the preparatory work has been done, this procedure shows how to actually build a part.

a. On process computer, press ENTER - MAIN MENU is displayed.

b. Select option 4 (Supervisor) and press ENTER.

c. In response to Part Prefix prompt:, type name of part files and press ENTER. This causes on the laser to start tracing the first layer. Verify that SHUTTER OPEN and LASER ON indicators on operator control panel are illuminated.

d. Watch the first layers form.

1. Is the part centered on the elevator platform?

2. Do the first layers stick to the platform?

3. If not, abort the run and correct the problem.

3.5.5.6 Post Processing of SLA-1 Parts. This procedure shows how to remove the finished part from the vat, drain the part, cure and dry the part, and remove supports.

a. Updip and Preliminary Draining.

1. On process computer, press ENTER - main menu is displayed.

2. Select Utilities menu and press ENTER.

3. Select Z-Stage Mover and press ENTER.

4. Very slowly toggle ↑ arrow until elevator is raised to 1 inch above edge of reaction vat level. Do not raise part quickly or distortion could occur.

5. Wait approximately 15 minutes to allow excess resin to drip from part.

b. Part/Platform Removal.

1. Place a pad of absorptive material on the special drain tray.

2. Slip the drain tray under the elevator platform and rest it on the shelves on either side of the vat.

3. Toggle ↓ arrow on keyboard to lower the elevator platform until it is about 1/4 inch above the absorptive pad.

4. Twist one of the elevator shaft knobs one turn ccw. This will unscrew a threaded rod inside the elevator shaft from a threaded hole in one side of the elevator platform and partially release the platform.

5. Repeat step (d) with the opposite elevator shaft knob.

6. Repeat steps (d) and (e) alternately until the platform drops free from the shafts and falls a fraction of an inch to rest on the absorptive pad.

7. Raise the elevator shafts, if necessary, with the ↑ arrow on keyboard.

8. Remove the drain tray, platform, and attached part from the vat chamber. Keep the platform horizontal if possible to avoid putting side stresses on the green (uncured) part.

c. Oven Draining.

1. Place elevator platform and part in an oven.

2. Set temperature to 80 to 90°C and wait 1 hour.

3. Carefully wipe off excess liquid resin which adheres to up and down-facing surfaces with cotton swabs.

d. Post Curing.

1. Place elevator platform and part in an ultraviolet flood oven.

2. Flood the part with ultraviolet light until it is dry and no longer tacky.

3. Use a fine-tooth saw and remove the part from the elevator platform by sawing through the bottom supports that attach the part to the platform. Protect the part from streses and inertial shocks during this procedure.

4. Clean all "sawdust" and fragments of support from the part before continuing.

5. Turn part upside down (or rest part on its side if this is not possible) and repeat steps 1 and 2.



systems, inc. COMPANY CONFIDENTAL

e. Platform Replacement.

1. Scrape away any dry resin that is still sticking to elevator platform after removal of part. If may be necessary to tap out the treaded holes in the platform with a #10-32 tap.

2. Place the empty platform on the drain tray.

3. Place the drain tray in the SLA-1 vat chamber resting on the shelves and centered as well as possible over the vat.

4. Very slowly toggle ↓ arrow on keyboard to lower the elevator shafts until the threaded rods are very close to the platform. Avoid running the shafts into the platform or tray as this can ruin the threads on the shaft.

5. Adjust the tray and platform so that the threaded holes in the platform are exactly aligned beneath the threaded rods.

6. Slowly toggle ↓ arrow on keyboard until the threaded rods contact the threaded holes gently.

7. Twist one of the elevator shaft knobs one turn cw. This will rotate the threaded rod inside the elevator shaft and cause it to engage the threaded hole in the panel.

8. Repeat step 7 with the opposite elevator shaft knob.

9. Repeats steps 7 and 8 alternately until the platform is lifted off of the drain tray and is in firm contact with the bottom of the elevator shafts.

10. Snug up the elevator shaft knobs to secure the panel to the shafts. Do not overtighten as this can fracture the interior threaded rod.

11. Toggle the ↑ arrow on keyboard to raise the elevator.

12. Remove the drain tray.

f. Removing Supports and Finishing.

1. Carefully cut away supports with side cutting pliers.

2. Carefully smooth off rough surfaces with appropriate files.

3. Surface finish as required.

3.5.5.7 Shutdown Procedures.

a. Set OVEN LIGHT switch to off (down). Verify that OVEN LIGHT indicator extinguishes.

b. Set POWER ON switch to off (down). Verify that POWER ON and other indicators extinguish.

3.5.5.8 How to Create and Use Working Curves. The degree to which liquid plastic can be solidified is determined by three factors: (1) the type of resin used, (2) laser power, and (3) degree of laser focus. By creating a working curve, the operator can adjust the laser drawing speed to compensate for variations in these three factors. Therefore, a new working curve must be prepare each time a new batch of resin is used or there is a significant loss of laser power as shown on the Part Log. The working curve is then used to change the step period (laser drawing speed) and step size in the default parameter and layer control files.

a. Toggle ↓ arrow on keyboard to lower elevator platform to 1 inch below resin surface. The banjo part used to create working curves will be prepared free floating on the resin surface.

b. On process computer, press ENTER - main menu is displayed.

c. Select Utilities menu an press ENTER.

d. Select Banjo and press ENTER. Follow the menu to input the largest step period (SP) to be used. The SLA-1 will prepare the banjo part in the vat.

e. After banjo part is completed, drain and cure it (subparagraph 3.5.5.6).

f. Use a microscope and measure horizontal width of each string.

g. Cut banjo part sideways and use microscope to measure thickness (depth) of each string.

h. Plot height and width values on a working curve (sample in Figure 3-2) at the step periods chosen (example - 40, 80, 160, 320, and 640). The lowest step period will create the thinnest banjo string and the highest step period will create the thickest banjo string.

i. Another banjo can be created with different step periods to extend the range of the working curve.

j. Connect the five or more points to form both working curves.

k. The working curve can now be used to select the step period and step size for each slice.

l. Insert the chosen step period and step size into the default parameter files (paragraph 3.5.5.2).

systems, inc.

COMPANY CONFIDENTAL

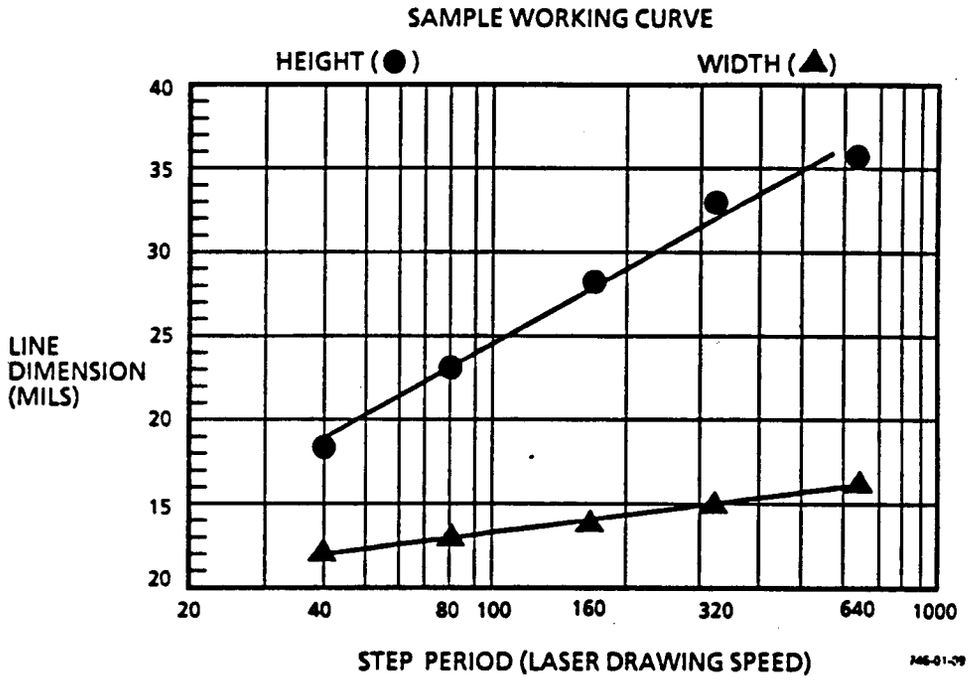


Figure 3-2. Sample Working Curve



systems, inc.

COMPANY CONFIDENTAL

SECTION IV

TROUBLESHOOTING

4.1 INTRODUCTION

This section contains troubleshooting procedures to identify and correct operation errors and selected hardware failures. Fault conditions not covered in this section shall be corrected by qualified maintenance technicians using the SLA-1 Service Manual.

4.2 TROUBLESHOOTING PROCEDURES

Beta SLA-1 troubleshooting procedures are listed in Table 4-1.

Table 4-1. Troubleshooting Procedures

Problem	Probable Cause	Remedy

SECTION V

OPERATOR MAINTENANCE INSTRUCTIONS

5.1 INTRODUCTION

This section contains procedures for routine maintenance to be performed by the SLA-1 operator. Included are procedures to clean up resin spills, add resin to vat, replace vat, replace air filter, and clean optical components. Maintenance requirements not covered in this section shall be performed by qualified technicians in accordance with SLA-1 Service Manual.

5.2 MATERIALS AND EQUIPMENT

Recommended maintenance materials and equipment are listed in Table 5-1. Equivalent items may be used.

Table 5-1. Maintenance Materials and Equipment

Nomenclature	Model/Part No.	Source
Gloves, chemical resistant		Commercially available
Glasses, safety		Commercially available
Towels, disposable		Commercially available
Container, hazardous materials disposal		Commercially available
Isopropanol		Commercially available
Clothing, protective		Commercially available
Respiratory equipment, NIOSH/MSHA approved		Commercially available
Absorbing material (sawdust, clay, diatomaceous earth, or activated charcoal)		Commercially available
Resin, UV curable		Commercially available
Compressed air		Commercially available
Methanol, reagent-grade		Commercially available
Glass		Commercially available
Lens tissue		Commercially available
Hemostat (or clamping tweezers)		Commercially available
Cotton swabs		Commercially available
Filter, air		Commercially available
Lamp, chamber		Commercially available
Lamp, indicator		Commercially available



systems, inc.

COMPANY CONFIDENTAL

5.3 RESIN CLEANING/REPLENISHMENT PROCEDURES

5.3.1 Cleanup of Small Resin Spills.

WARNING

UV CURABLE RESINS MAY CAUSE EYE AND SKIN BURNS. REPEATED OR PROLONGED SKIN CONTACT MAY CAUSE SENSITIZATION. VAPOR MAY BE HARMFUL.

a. Read the safety precautions and first aid instructions at the front of this manual.

b. Put on chemically resistant gloves and safety glasses.

c. Remove the spilled resin using disposable towels.

d. Seal contaminated towels and other contaminated articles in a marked container suitable for the disposal of hazardous materials.

NOTE

Fully cured resins present no safety or health related hazards. An alter-native means of disposing lightly contaminated articles is to ultraviolet or heat-cure the affected areas and dispose of as nonhazardous waste.

e. Clean the spill area with isopropanol followed by a thorough washing with soap and water.

f. Remove gloves and seal in disposal container. Dispose of container in accordance with applicable hazardous waste laws and regulations.

g. Wash hands thoroughly with soap and water.



systems, inc.

COMPANY CONFIDENTAL

5.3.2 Cleanup of Large Resin Spills.

WARNING

UV CURABLE RESINS MAY CAUSE EYE AND SKIN BURNS. REPEATED OR PROLONGED SKIN CONTACT MAY CAUSE SENSITIZATION. VAPOR MAY BE HARMFUL.

a. Immediately isolate the spill area.

b. Read the safety precautions and first aid instructions at the front of this manual.

c. Put on protective clothing, chemically resistant gloves and safety glasses. If the spill area is not well ventilated, wear NIOSH/MSHA approved respiratory equipment.

d. Cover the spill with absorbing material such as sawdust, clay, diatomaceous earth, or activated charcoal.

e. When the absorbant is saturated, seal it in a marked container suitable for the disposal of hazardous materials.

f. Clean the spill area with isopropanol followed by a thorough washing with soap and water.

g. Wipe resin off protective clothing with clean disposable towels.

i. Remove protective clothing in the following order: boots, gloves and suit. After removing gloves, use disposable cloths to protect hands.

j. Place contaminated articles in the disposal container. Dispose of container in accordance with applicable hazardous waste laws and regulations.

k. Shower with soap and cool water.

5.3.3 Vat Replenishment. The reaction vat contains spillways on both sides to handle resin overflow. The overflowed resin then flows through the overflow tubes, through the drain valve, and settles in the collection bottle. Do not reuse resin after an overflow. If resin level in vat is low, replenish as follows:

a. Open chamber door.

b. Slowly pour new resin into vat. Make sure that new resin is the same type as the resin in the vat. The part will not build properly if different resins are mixed in the vat.

c. Wait 1/2 hour for resin level to settle before resuming operation.

d. Close chamber door.

5.4 OPTICS CLEANING (Figure 5-1)

The SLA-1 optics must be cleaned when there is obvious contamination (fingerprints, dust, haze) of the optical surfaces, or when the laser power measured at the resin surface is less than the power measured at the laser by more than a few milliwatts. If laser power loss is present, the optical alignment should be checked. If the system is aligned as well as possible and still there is a significant difference in power from the laser to the resin surface, optics cleaning is required.

5.4.1 Equipment Required.

a. A source of clean, compressed air designed for optics cleaning.

b. A small amount of clean reagent-grade methanol in a glass (not plastic) bottle.

c. A supply of clean lens tissue. (Conventional face tissues leave lint on optical surfaces and are not recommended. Industrial wiping tissues are designed to be abrasive, and should be strictly avoided.)

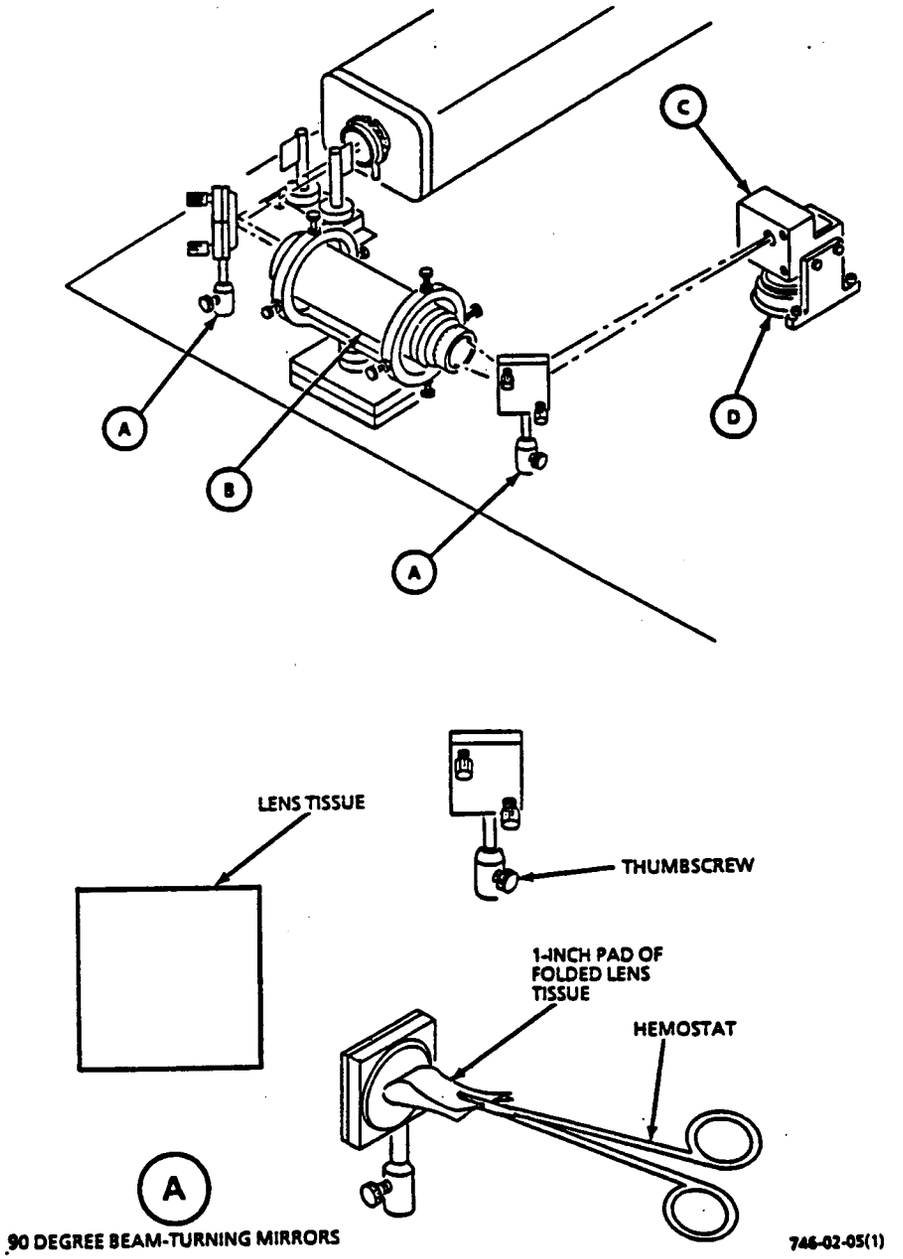


Figure 5-1. Recommended Optics Cleaning Technique (Sheet 1 of 2)

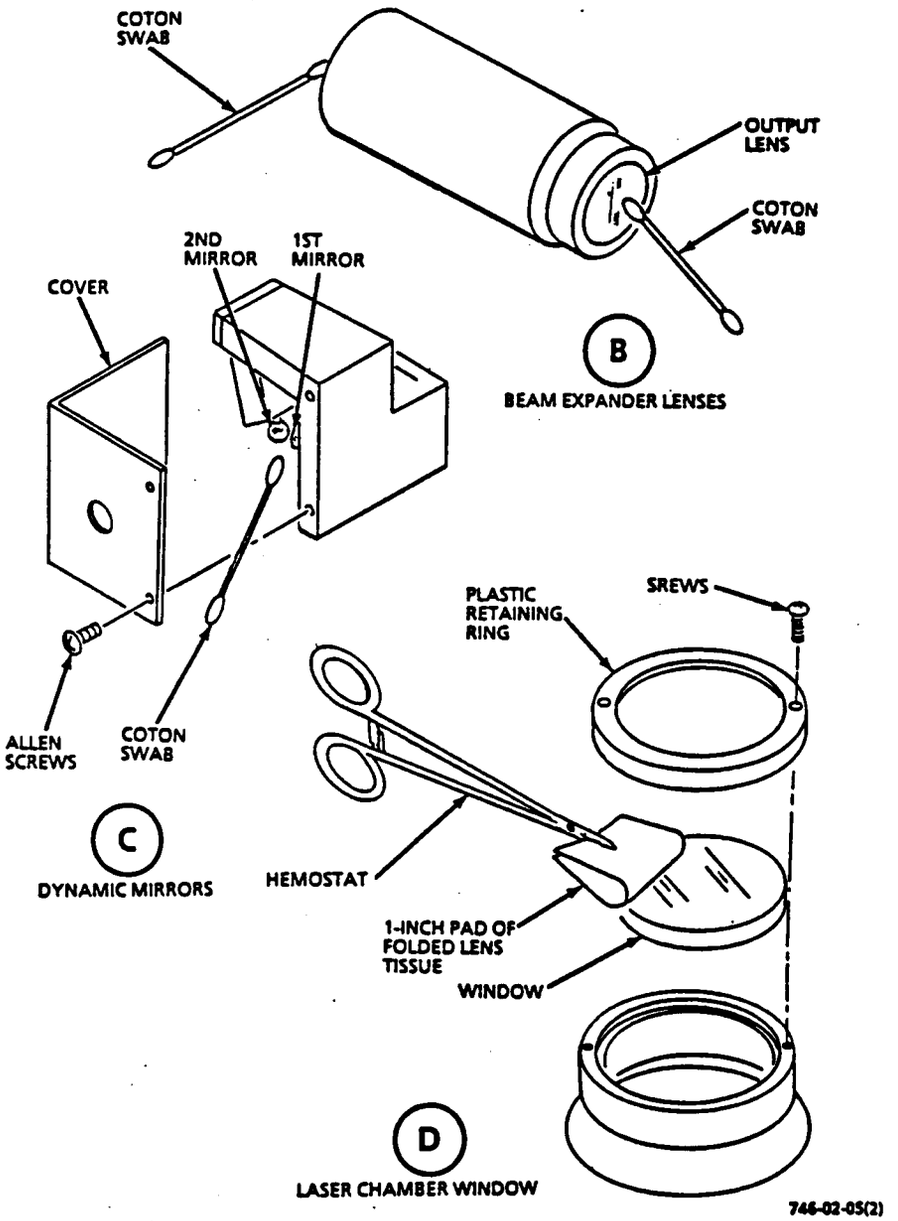


Figure 5-1. Recommended Optics Cleaning Technique (Sheet 2 of 2)

d. A hemostat or pair of clamping tweezers.

e. A supply of special cotton swabs designed for optics cleaning. (These have a wooden stick with the cotton heads attached without glue. Plastic "sticks" and glue can be dissolved by the methanol and their residue contaminates the optical surfaces.)

f. Freshly scrubbed, clean, dry hands.

5.4.2 Care and Handling of Optics. All of the optics in the SLA-1 are coated with precise dielectric coatings, either to avoid unwanted reflections from the surfaces of the transmissive optics or to achieve efficient reflections from the mirrors. These coatings are fragile and easy to damage. Even faint fingerprints will damage the coatings, so extreme care must be taken to never touch the optical surfaces with fingers or with anything except carefully handled lens tissue or special cotton swabs. The optics must be handled only by the edges of any of the optics, or by the uncoated backs of the reflective optics.

5.4.3 Laser Brewster Windows. Refer to HeCd Laser Service Manual to clean laser brewster windows.

5.4.4 Laser Resinator Mirrors. Refer to HeCd Laser Service Manual to clean laser resinator mirrors.

5.4.5 90 Degree Beam-Turning Mirrors. (Figure 5-1A) The beam-turning mirrors can be removed and cleaned after their protective covers/shields have been removed. Turn the thumbscrew locking the mirror mount post into the post-holder ccw and gently lift the post and mirror mount of the post-holder. If desired, the mirrors and their immediate black-anodized holders can be separated from the mount by loosening the thumbscrew which retains the mirror holder to the mirror mount.

a. Carefully remove dust and lint from the mirror surface using clean compressed air. If using an aerosol "duster", be certain that the can is always held upright or aerosol propellant will be sprayed onto and contaminate the optical surface.



systems, inc. COMPANY CONFIDENTAL

b. Wash hands thoroughly and fold a new sheet of lens tissue in halves repeatedly to achieve a "pad" of lens tissue about 1-inch square, and several thicknesses of tissue thick. Grasp this pad along one edge with a hemostat or locking tweezers.

c. Apply a few drops of reagent-grade methanol to the pad and shake off any excess.

d. Apply the side of the pad to one edge of the surface of the mirror. Carefully draw the pad across the surface one time only using some pressure on the hemostat. Allow the "tail" of the pad to follow the leading edge of the pad and drag across the optical surface. Immediately discard this piece of lens tissue.

e. Allow the small film of methanol remaining on the optic to evaporate naturally. Do not wipe or blow on the optic to assist evaporation.

f. Carefully examine the optic by looking at a glancing reflection of a bright area. If the optic still is not clean, choose a new sheet of lens tissue and repeat steps b through f until satisfactory results are achieved.

g. If it is desired to clean the beam expander lenses as well, do not replace the 90 degree beam-turning mirrors and mounts until the lenses have been cleaned as explained below.

5.4.6 Beam Expander Lenses. (Figure 5-1B) Do not remove the lenses from the beam expander mount. The mount is difficult to realign and should not be removed.

a. In order to clean the beam expander optics in place, the 90 degree beam-turning mirrors and mounts should be removed as explained above and carefully set aside.

b. Apply a few drops of reagent grade methanol to an unused special cotton swab and shake or flick off any excess.

c. To clean the input expanding (small) lens:

1. Carefully insert the soft tip of the swab into the input aperture until the surface of the lens is encountered. Use slight pressure and rotate the swab as it contacts the lens. Remove the swab and immediately discard it.

2. Repeat steps b and c.1 with a new, much "drier" swab from which almost all of the methanol has been shaken out.

3. Repeat step c.1 with a new, completely dry cotton swab.

d. To clean the output converging (large) lens:

1. Apply a few drops of reagent grade methanol to an unused special cotton swab and shake or flick off any excess.

2. Use soft pressure with the soft tip of the swab and begin in the center of the lens. Wipe the lens with an expanding spiral pattern until the outer edge is wiped once. Immediately discard the swab.

3. Repeat step d.2 with a new, much "drier" swab from which almost all of the methanol has been shaken out.

4. Repeat step d.1 with a new, completely dry cotton swab.

5. Now carefully examine the lens by looking at a reflection of a bright area. If the lens still is not clean, choose a new special cotton swab and repeat steps d.1 through d.5 until satisfactory results are achieved.

e. Check the cleaning job by viewing the expanded beam.

1. Replace the input 90 degree beam-turning mirror.

2. Put on UV blocking glasses or goggles.

3. Defeat the interlock shutter by using the interlock-defeat jumper and realign the turning mirror so that the laser beam passes directly through the center of the beam expander.

4. Place a fluorescing piece of paper in front of the output lens and watch the expanded beam image carefully while adjusting the input turning mirror slightly to move the beam around through the beam expander. Any discontinuties in the beam pattern which do not move with the beam as it moves are likely to be caused by blemishes on the beam expander optics. If any are observed, repeat paragraph 5.3.5.6 entirely.

f. Replace the output beam-turning mirror and mount it in the post holder.

5.4.7 Galvanometer-Driven Dynamic Mirrors. (Figure 5-1C)

CAUTION

The galvanometer-driven dynamic mirrors are very light, thin, carefully balanced, and fragile. Breaking a dynamic mirror is easy to do and will involve a major replacement of the dynamic mirror system, and realignment of the SLA-1 optics. Therefore, be especially careful and gentle when cleaning the dynamic mirrors.

a. Remove the protective cover from the dynamic mirrors by loosening the two machine screws which attach it to the dynamic mirror bracket. Be certain not to confuse them with the machine screws which mount the entire dynamic mirror bracket. Do not remove the dynamic mirror bracket from its mount, nor the mount from the optical plate. Detaching and remounting the dynamic mirrors could necessitate a major optical realignment, and is not recommended.

b. Apply a few drops of reagent grade methanol to an unused special cotton swab and shake or flick off any excess.

c. Hold the stick of the swab between the fingers and carefully bring the side of the swab into contact with one end of the front surface of one of the dynamic mirrors.

d. Use gentle pressure and wipe the swab along the long axis of the mirror once while rotating the swab in the opposite direction. Immediately discard the swab.

e. Allow the small film of methanol remaining on the optic to evaporate naturally. Do not wipe or blow on the optic to assist evaporation.

f. Repeat steps c through e for the other dynamic mirror.

g. Now carefully examine the optics by looking at a glancing reflection of a bright area. If the optics still are not clean, choose a new cotton swab, and repeat steps c through g until satisfactory results are achieved.

5.4.8 Laser Chamber Window. (Figure 5-1D)

a. Open chamber door.

b. Remove two screws from inside chamber and remove window mount.

c. Remove two screws and remove plastic retaining ring from window mount.

d. Wash hands thoroughly, touch only the edges of the window, and remove the window from the mount.

e. Carefully remove dust and lint from the surface using clean compressed air. If using an aerosol "duster", be certain that the can is always held upright or aerosol propellant will be sprayed onto and contaminate the optical surface.

f. Fold a new sheet of lens tissue in halves repeatedly to achieve a "pad" of lens tissue about 1-inch square, and several thicknesses of tissue thick. Grasp this pad along one edge with the hemostat or locking tweezers.

g. Apply a few drops of reagent grade methanol to the pad and shake off any excess.

h. Now apply the side of the pad to new edge of the surface of the window as shown in. Carefully draw the pad across the surface one time only using some pressure on the thermostat. Allow the "tail" of the pad to follow the leading edge of the pad and drag across the optical surface. This will clean one section of the surface of the large window. Immediately discard this piece of lens tissue.

i. Use a new pile of lens tissue each time and repeat steps d through f until the entire surface has been cleaned.

j. Allow the small film of methanol remaining on the window to evaporate naturally. Do not wipe or blow on the window to assist evaporation.

k. Carefully examine the window by looking at a glancing reflection of a bright area. If the window still is not clean, choose a new sheet of lens tissue and repeat steps d through i until satisfactory results are achieved.

l. Repeat steps d through i to clean the other side of the window.

m. Carefully replace the window in the mount.

n. Install plastic retaining ring on window mount and secure with two screws.

o. Install window mount in chamber window aperture and secure with two screws.

p. Close chamber door.

5.5 REPLACEMENT PROCEDURES

5.5.1 Air Filter Replacement. (Figure 5-2) Replace the air filter every 6 months.

a. Open chamber door.

b. Place lit on vat.

c. Remove two screws and pull air filter with clamp attached straight out from chamber.

d. Install new air filter in chamber and secure with two screws.

e. Remove lid from vat.

f. Close chamber.

5.5.2 Chamber Lamp.

a. Set OVEN LAMP switch to off.

b. Open chamber door.

c. Grasp chamber lamp at both ends and pull straight down to disengage lamp.

d. Install new lamp by pushing both ends to engage the prongs.

e. Close chamber door.

f. Set OVEN LAMP switch to on. Verity that chamber lamp is illuminated.

systems, inc.

COMPANY CONFIDENTAL

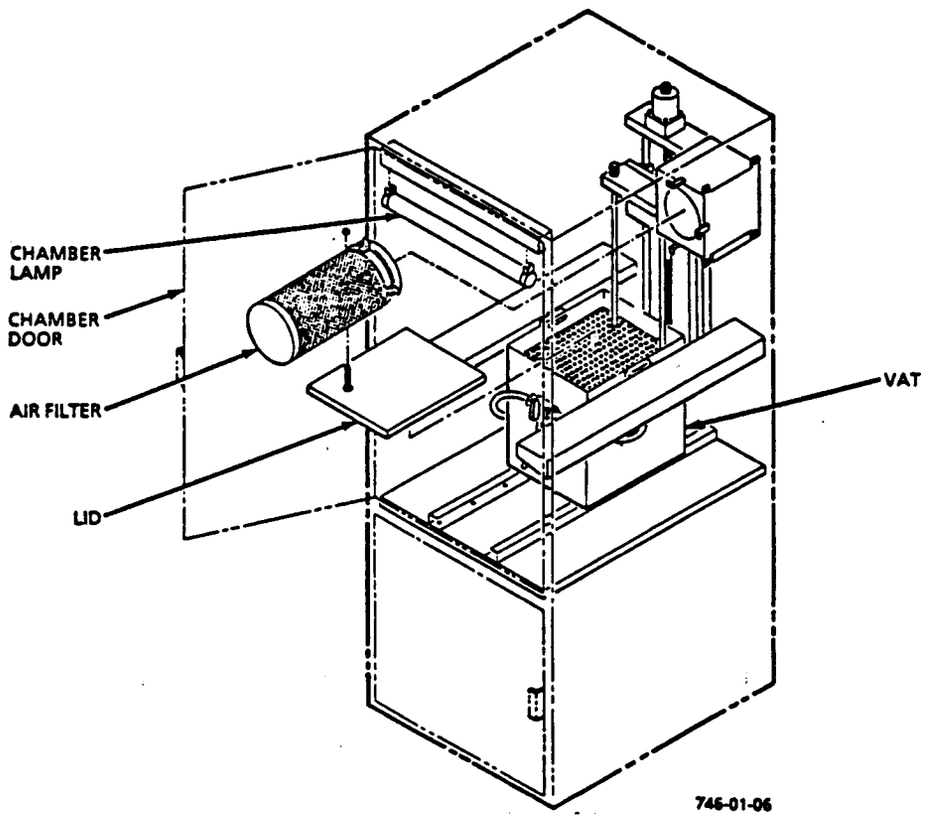


Figure 5-2. Air Filter Replacement



systems, inc.

COMPANY CONFIDENTAL

5.5.3 Control Panel Lamps.

a. Place a small flat blade screwdriver on bottom of plastic lens and push up to disengage lens.

b. Grasp plastic tab with screwdriver and pry lamp forward to remove.

c. Install new lamp in plastic tab.

d. Install plastic lens. Verify that lamp illuminates during normal operating sequence.



systems, inc.

COMPANY CONFIDENTAL

GLOSSARY

The following terms are used in the stereolithography process:

60/120 ANGLE HATCH	This is a type of crosshatching pattern which supplements the standard X and Y hatching. See software manual.
BANJO	This is a part which when built and measured provides line height and line width data for the working curve.
BASES (SUPPORTS)	These are the structures generated by the CAD which provide structural support to the actual part when being built. (See webs).
BEAM PROFILE	This is the spatial distribution of laser beam energy.
BORDER (BOUNDARY)	The border is a block of vectors defining the walls of a sliced layer of a part.
CAD	Computer aided design.
CENTERING	This is a SLICE routine which automatically centers the part in space. This can only be done if the part is defined by only one STL file. See software manual.
CLIFF	This is a software program which is used primarily in BASIC programming. It can also be used to move mirrors using direct commands from the DOS shell by transmitting data to STEREO.
CRITICAL AREA	This is an area within the part which has coordinates defined in a text file prior to merging. The area can have special attributes such as riveting. See software manual.



systems, inc.

COMPANY CONFIDENTAL

CROSSHATCH	This is a general interior vector type providing structural integrity to walls. The pattern used is defined during slicing. See software manual.
CURL	This is an effect sometimes encountered during part creation which can cause part inaccuracy.
DIP ACCELERATION	This is a part-building variable which defines the speed. of elevator dipping. It can be modified on a layer by layer basis if needed.
DIP DELAY	This is a part-building variable defining the delay between the start of a dip routine and the start of next layer calculations (and laser movement). It can be modified on a layer by layer basis.
DIP DEPTH	This a part-building variable which defines the distance the elevator moves downward during a dipping sequence.
DIP SPEED	This is a part-building variable which defines the maximum speed that the elevator attains.
DRAWING SPEED	Laser drawing speed is defined by the Supervisor variables Step Period and Step Size. It is varied depending on layer thickness, type of resin, and laser power. The drawing speed to be used is typically generated by use of the working curve.
DYNAMIC MIRRORS	These are the galvanometer vector scanning mirrors which are controlled by SLA-1 software. Laser beam movement is defined by the rotation of these mirrors.
ELEVATOR	The elevator is the vertically moving device to which the elevator platform and part are attached.

ETHERNET	This is file transfer software system. It allows for easy movement of large files.
FOOTPRINT	This is the bottom of the supports which adheres directly to the elevator platform.
GREEN PART	This is the laser cured part which has not undergone final post curing.
HATCH SPACING	This is the distance between crosshatching which is defined during slicing (see crosshatch).
HeCd	Helium Cadmium.
L FILE	This is a merge generated control file which contains all the layer by layer vector block identification information. Individual layer parameters can be modified in the L file.
LASER	This is the laser controller software. If is also a device which provides the light energy required to polymerize the liquid photopolymer.
LAYER THICKNESS	This is the layer to layer dipping distance. It can be either one value for the whole part or changed many times throughout the part (see variable layer thickness).
LEVELING	This is the time and temperature-dependent process where the resin settles to a flat surface after being distributed by dipping. The time allowed for leveling is defined by the dip delay variable.
LINE HEIGHT	This the vertical thickness of a laser cured plastic line. It is variable depending on drawing speed and laser power/focus.



systems, inc.

COMPANY CONFIDENTAL

LINE WIDTH	This is the width of a laser cured plastic line. It is variable depending on drawing speed and laser power/focus.
MERGE	This is a software program which takes the individual sliced files for a part and combines them. It generates the L and V files which Supervisor uses to make the part.
MIA	This is the minimum intersect angle and is used during slicing to delete hatch vectors which run parallel to layer boundaries. See software manual.
MODULUS	This is a physical attribute of a material which defines it's overall toughness.
MONOMER	This is a chemical species which is generally low in molecular weight and is used as a building block for preparing polymers.
MSA	This is the minimum surface angle and is used during slicing. See software manual.
MSHA	Mine Safety and Health Administration.
NIOSH	National Institute for Occuptational Safety and Health.
PHIGS FORMAT	This is the software program that defines the CAD surfaces by using triangles.
PHOTOINITIATOR	This is a chemical which transforms the laser energy into chemical energy and initiates the polymerization process.
PHOTOPOLYMER	This is a polymer which is formed using light as the energy source.

POST CURE	This is the process used to cure a green part. The post cure can be either ultraviolet light induced or thermal induced.
POT LIFE	This is the useful life expectancy of a pot of chemical and depends on the chemical stability and other factors.
PRIMARY RADICAL	This is the initial radical species formed when laser light is absorbed by the photoinitiator. The primary radicals initiate the polymerization process.
RADIAL CROSSHATCH	This is a specific type of crosshatch which generally provides the best overall strength and support (see crosshatch).
RADIOMETER	This is a device which allows for the measurement of the laser output power.
RESIN	This is the liquid photopolymer.
RIVET	This is a part making process which can be used in critical areas which may be prone to stress related inaccuracies.
SCALE FACTOR	This is a Supervisor variable which can be used to scale the XY space either larger or smaller. It does not affect the vertical dimension.
SENSITIZATION	This is an allergic response that some individual may obtain after repeated skin contract with a given chemical.
SKIN (SKIN FILL)	This is the coating of a horizontal (flat) or near horizontal (flat) area of the part.
SLA	Stereo-lithography Apparatus



systems, inc.

COMPANY CONFIDENTAL

SLICE	This is the software that transforms the three-dimensional CAD designed part into a series of two-dimensional layers (slices).
SMALLEY	This is a stress relieving (decoupling) structure designed on the CAD.
STEP PERIOD	This is a Supervisor variable which helps to define the laser drawing speed. Increasing the step period slows the speed (increases the plastic line height and width).
STEREO	This is a memory resident portion of the laser controller software.
STL FILE	This is the PHIGS format CAD file used as input for slicing.
SUPERVISOR	This is the software which supervises all the passing of variables and data to drive the mirrors and move the Z stage up and down during part building.
TENSILE STRENGTH	This is an attribute of a material which defines the energy required to stretch it.
TRAPPED VOLUMES	These are areas of the part from which resin cannot drain during dipping.
USER INTERFACE	This is the menu driven software which is used to control and run slicing programs.
.V FILE	This is a merge generated file which contains all the layer by layer vector information. See software manual.



systems, inc.

COMPANY CONFIDENTAL

VARIABLE LAYER THICKNESS	This is the process tool which allows use of differing dipping depths and layer thicknesses in order to achieve improved strength or accuracy. It is controlled within Slice.
WEB	This is a type of support structure designed by the CAD designer which can provide additional strength or support as needed.
WORKING CURVE	This is the plotted line height and line width data provided from Banjotop. It is used in with laser power to obtain drawing speed information.



systems, inc.

NOT TO BE DISCLOSED

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION RELATING TO STEREOLITHOGRAPHY PROCESSES AND EQUIPMENT WHICH ARE THE EXCLUSIVE PROPERTY OF 3D SYSTEMS, INC. THIS INFORMATION SHALL NOT BE DUPLICATED, USED OR DISCLOSED – IN WHOLE OR IN PART – FOR ANY PURPOSE OTHER THAN TRAINING AS AUTHORIZED IN WRITING BY 3D SYSTEMS, INC. ALL REASONABLE PRECAUTIONS SHALL BE TAKEN TO ENSURE COMPLIANCE WITH THESE CONDITIONS.

THE POSSESSION OF THIS DOCUMENT CONVEYS NO PROPERTY RIGHT OR LICENSE IN THE INFORMATION NOR IN ANY MATTER DISCLOSED THEREIN.

SLA-1

BETA SITE

STEREOLITHOGRAPHY

SYSTEM

SERVICE MANUAL

TABLE OF CONTENTS

INTRODUCTION

TROUBLESHOOTING

COMPONENT REPLACEMENT

ALIGNMENT

SPARE PARTS LISTS

WIRING DIAGRAM

3D Systems Incorporated
12847 Arroyo Street
Sylmar, California 91342
(818) 898-1533 • FAX (818) 361-5484

NOVEMBER 1987



systems, inc.

COMPANY CONFIDENTAL

Safety Precautions

WARNING
INVISIBLE LASER RADIATION

UV LASER RADIATION MAY DAMAGE EYE TISSUE. AVOID DIRECT EXPOSURE TO LASER BEAM. WEAR UV BLOCKING SAFETY GLASSES.

WARNING
HIGH VOLTAGE

HAZARDOUS VOLTAGES ARE ACCESSIBLE WITH MAINTENANCE PANELS REMOVED. FOLLOW SAFE WORK PRACTICES.

WARNING
CHEMICAL HAZARDS

UV CURABLE RESINS MAY CAUSE EYE AND SKIN BURNS. REPEATED OR PROLONGED SKIN CONTACT MAY CAUSE SENSITIZATION. VAPOR MAY BE HARMFUL.

1



systems, inc.

COMPANY CONFIDENTAL

Observe the following safety precautions when cleaning spilled resin or handling resin containers.

- Keep away from heat, sparks and flame. Protect from sunlight and fluorescent light. Closed containers may rupture/explode when exposed to extreme heat. Use National Fire Protection Association Class B extinguishers (carbon dioxide, dry chemical or foam).
- Use only with adequate ventilation. Avoid breathing vapors or spray mist.
- Wear safety glasses.
- Wear chemically resistant gloves and protective clothing. Wash thoroughly after handling and before eating, smoking or using toilet facilities.
- Wear an NIOSH/MSHA approved respirator or ensure adequate ventilation when sanding or cutting cured objects.

FIRST AID FOR CHEMICAL ACCIDENTS

Skin Contact. Wash thoroughly with soap and water. Remove contaminated clothing and shoes immediately. If skin is irritated, get medical attention. Wash clothing before reuse and discard contaminated shoes.

Eye Contact. Flush immediately with large amounts of water for 15 minutes and avoid sunlight, flourescent light, or other ultraviolet light. Get medical attention.

Inhalation. Remove victim to fresh air. Give artificial respiration or cardio-pulminary resuscitation if required. If breathing is difficult, give oxygen. Get medical attention.

11



systems, inc.

COMPANY CONFIDENTAL

TABLE OF CONTENTS

Section/Paragraph	Title	Page
	INTRODUCTION	v
I	TROUBLESHOOTING	1-1
1.1	Introduction	1-1
1.2	How to Use this Section	1-1
1.3	Test Points	1-3
II	COMPONENT REPLACEMENT	2-1
2.1	Introduction	2-1
2.2	Tools	2-1
2.3	Sice Computer	2-2
2.3.1	Monitor	2-2
2.3.2	Keyboard	2-2
2.3.3	Slice Computer	2-4
2.4	Electronic Cabinet Assembly	2-4
2.4.1	Monitor	2-4
2.4.2	Keyboard	2-7
2.4.3	Process Computer	2-7
2.4.4	Laser Power Supply	2-8
2.4.5	Elevator Driver	2-9
2.4.6	AC Power Distribution Panel	2-10
2.4.7	Dynamic Mirrors Driver	2-11
2.5	Optics Assembly	2-12
2.5.1	Laser	2-12
2.5.2	Shutters	2-15
2.5.3	Beam-Turning Mirrors	2-15
2.5.4	Beam Expander	2-16
2.5.5	Dynamic Mirrors	2-16
2.5.6	Interlock Switches	2-17
2.5.6.1	Laser Cover	2-17
2.5.6.2	Optics Cover	2-18
2.6	Chamber Assembly	2-18
2.6.1	Heater/Fan	2-18
2.6.2	Reaction Vat	2-20
2.6.3	Elevator	2-21
2.6.4	Platform	2-21
2.6.5	Beam Profiler	2-22
II	ALIGNMENT	3-1
3.1	Introduction	3-1
3.2	Tools and Equipment	3-1
3.3	Laser Resonator	3-2
3.4	Optics	3-4
3.5	Chamber	3-7

TABLE OF CONTENTS - CONT

Section/Paragraph	Title	Page
3.5.1	Vertical Alignment of Elevator	3-7
3.5.2	Horizontal Alignment of Optics Plate .	3-9
IV	SPARE PARTS LIST	4-1
4.1	Introduction	4-1
4.2	To to Use this Section	4-1
4.3	Manufacturer's Names and Addresses . .	4-2
V	WIRE LIST AND DIAGRAM	5-1
5.1	Introduction	5-1
5.2	Wire List and Diagram Diagram	5-1

LIST OF FIGURES

Figure	Title	Page
2-1	Slice Computer Components	2-3
2-2	Electronic Cabinet Components	2-5
2-3	Optics Components	2-13
2-4	Chamber Components	2-19
3-1	Laser Resonator Alignment	3-3
3-2	Optics Alignment	3-5
3-3	Chamber Alignment	3-8
4-1	SLA-1 Stereolithrography System . . .	4-3
4-2	Electronic Cabinet Assembly	4-5
4-3	Optics Assembly	4-7
4-4	Chamber Assembly	4-9
5-1	SLA-1 Wiring Diagram	5-2

LIST OF TABLES

Figure	Title	Page
1-1	Fault Indication Chart	1-1
2-1	Replacement Tools	2-1
3-1	Alignment Toools and Equipment	3-1
4-1	Vendor Names and Addresses	4-2



systems, inc.

COMPANY CONFIDENTAL

INTRODUCTION

SCOPE

This manual contains service and corrective maintenance procedures for the SLA-1 Beta Site Stereolithography System. This manual is to be used by qualified maintenance technicians to identify and replace defective components. Refer to the SLA-1 User's Manual for operation and routine maintenance procedures.

ORGANIZATION

This manual is divided into five sections, as follows:

SECTION I. TROUBLESHOOTING - Diagnostic and corrective procedures.

SECTION II. COMPONENT REPLACEMENT - Component removal and installation procedures.

SECTION III. ALIGNMENT - Alignment of laser, optics, and chamber.

SECTION IV. SPARE PARTS LISTS - Exploded view illustrations and lists of authorized spare parts.

SECTION V. WIRING DIAGRAM - Wiring diagram to aid in troubleshooting and continuity tests.

OTHER FEATURES OF INTEREST

Title Page Index. On the right side of the title page you will find a section index. The black markers printed to the right of the text are keyed to similar markers on the first page of each referenced section. To locate a section, flip through the manual until you find a marker printed in the same location as the title page marker.

Warning Pages. Following the title page you will find a summary of critical warnings. You may be seriously injured if these warnings are not explicitly followed. Carefully read all warnings and first aid instructions prior to performance of operation or maintenance instructions.



systems, inc.

COMPANY CONFIDENTAL

REPORTING ERRORS AND RECOMMENDING IMPROVEMENTS

This manual was prepared for use by Beta Site maintenance personnel. It will be revised to incorporate recommended content and format changes. If you find any mistakes, or if you have any recommendations for improving the procedures, please let us know by sending a copy of the marked-up page(s) to:

Atten: Chick Lewis
3D Systems Inc.
12847 Arroyo St.
Sylmar, California 91342
(818) 898-1533 • FAX (818) 361-5484

systems, inc.

COMPANY CONFIDENTAL

SECTION I

TROUBLESHOOTING

1.1 INTRODUCTION

This section contains troubleshooting procedures to identify and correct selected hardware failures. Troubleshooting procedures for operation errors are covered in the SLA-1 User's Manual.

1.2 HOW TO USE THIS SECTION

a. Perform the appropriate troubleshooting procedure referenced in Table 1-1 in accordance with the following special instructions.

WARNING

HAZARDOUS VOLTAGES ARE ACCESSIBLE WITH MAINTENANCE PANELS REMOVED. FOLLOW SAFE WORK PRACTICES.

1. The troubleshooting procedures are presented in flow diagram format. Use of the diagrams is described in Figure 1-1.

2. Test point illustrations are provided following the flow diagrams (paragraph 1.3).

b. Repeat the test portion of the flow diagrams after any part is replaced.

Table 1-1. Fault Indication Chart

Fault Indication	Troubleshooting Procedure (Figure No.)



systems, inc.

COMPANY CONFIDENTAL

SECTION II

COMPONENT REPLACEMENT

2.1 INTRODUCTION

This section contains procedures to replace defective components. Each procedure is self-contained and is supported by an illustration keyed to the procedure in sequence of disassembly. Use the troubleshooting procedures in Section I to determine which components are defective.

WARNING

HAZARDOUS VOLTAGES ARE ACCESSIBLE WITH MAINTENANCE PANELS REMOVED. FOLLOW SAFE WORK PRACTICES.

2.2 TOOLS

Recommended replacement tools are listed in Table 2-1. Equivalent items may be used.

Table 2-1. Replacement Tools

Nomenclature	Part No.	Source
Screwdriver, flat blade, small		Commercialy available
Wrench set, allen		Commercialy available
Soldering iron, small		Commercialy available



systems, inc.

COMPANY CONFIDENTAL

2.3 SLICE COMPUTER (Figure 2-1)

2.3.1 Monitor.

a. Set power switch (1) on rear of monitor (2) to off.

b. Disconnect power cable (3) from monitor.

c. Use a small flat blade screwdriver to remove two connector screws and disconnect signal cable (4) from monitor.

d. Remove monitor.

e. Install new monitor.

f. Connect signal cable (4) to monitor and use a small flat blade screwdriver and install two connector screws.

g. Connect power cable (3) to monitor.

h. Set power switch (1) on rear of monitor to on.

2.3.2 Keyboard.

a. Set keyboard lockout keylock switch (5) on slice computer (6) to off.

b. Disconnect signal cable (7) from keyboard (8). The cable is terminated with a telephone jack.

c. Remove keyboard.

d. Install new keyboard.

e. Connect signal cable (7) to keyboard.

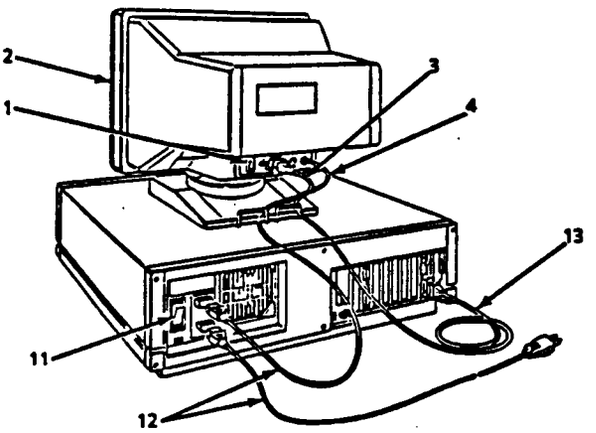
f. Set keyboard lockout keylock switch (5) on slice computer (6) to on.

systems, inc.

COMPANY CONFIDENTAL

LEGEND:

1. POWER SWITCH
2. MONITOR
3. POWER CABLE
4. SIGNAL CABLE
5. LOCKOUT SWITCH
6. SLICE COMPUTER
7. SIGNAL CABLE
8. KEYBOARD
9. DISKETTE
10. DISK DRIVE
11. POWER SWITCH
12 POWER CABLE (2 EA)
13. SIGNAL CABLE



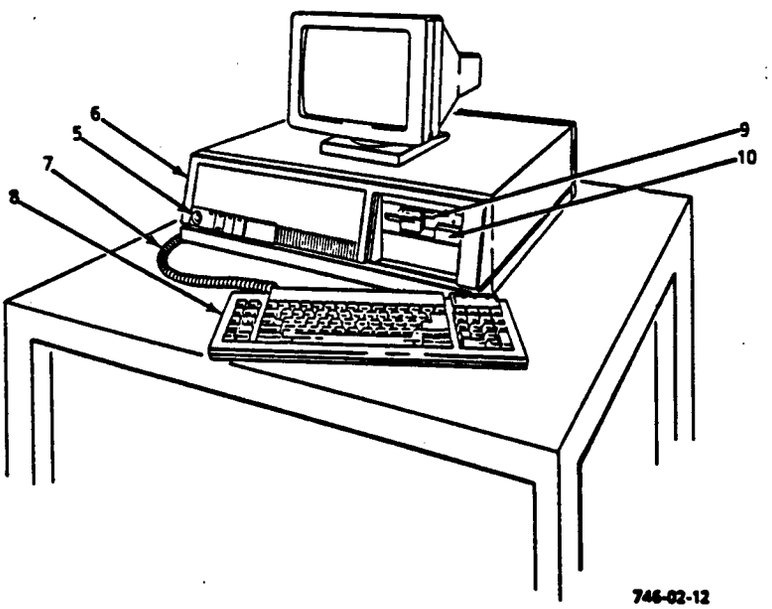


Figure 2-1. Slice Computer Components

2-3

2.3.3 Slice Computer.

CAUTION

Store the diskette in a safe place for protection.

a. Remove diskette (9) (if present) from disk drive (10).

b. Set keyboard lockout keylock switch (5) to off.

c. Set power switch (11) on rear panel to off.

d. Disconnect two power cables (12) (monitor power and ac input power) from rear panel.

e. Disconnect signal cable (7) (keyboard) from front panel.

f. Disconnect one signal cable (13) (monitor) and one coaxial cable (14) (Ethernet) from rear panel.

g. Remove slice computer (6).

h. Install new slice computer.

i. Connect signal cable (7) to front panel.

j. Connect one signal cable (13) and one coaxial cable (14) to rear panel.

k. Connect two power cables (12) to rear panel.

l. Set power switch (11) on rear panel to on.

m. Set keyboard lockout keylock switch (5) to on.

n. Install diskette (9) (if required) into disk drive (10).

2.4 ELECTRONIC CABINET ASSEMBLY (Figure 2-2)

2.4.1 Monitor.

a. Remove two screws (1) and remove cable access panel (2).

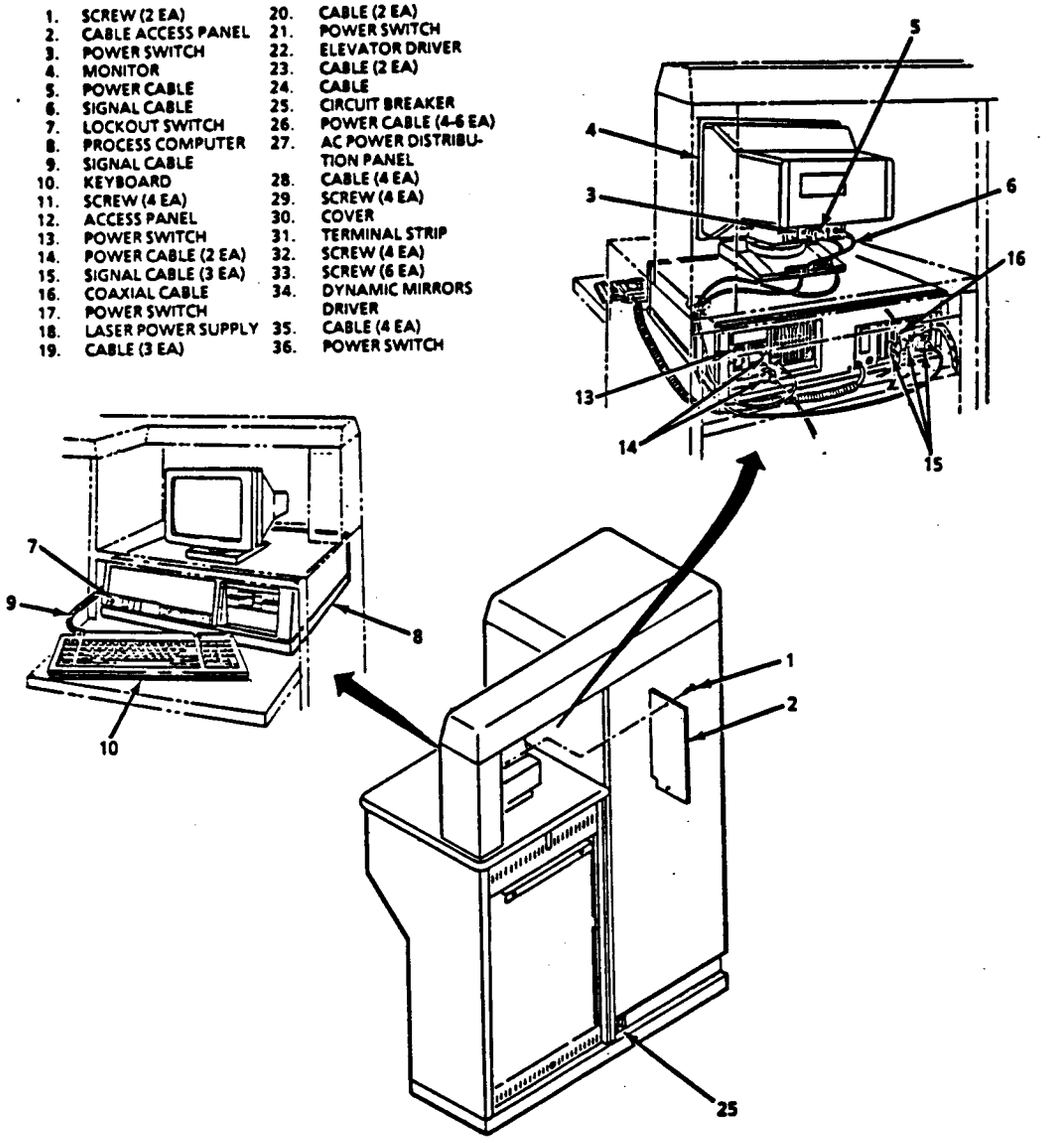
b. Set power switch (3) on rear of monitor (4) to off.

systems, inc.

COMPANY CONFIDENTAL

LEGEND:

1.	SCREW (2 EA)	20.	CABLE (2 EA)
2.	CABLE ACCESS PANEL	21.	POWER SWITCH
3.	POWER SWITCH	22.	ELEVATOR DRIVER
4.	MONITOR	23.	CABLE (2 EA)
5.	POWER CABLE	24.	CABLE
6.	SIGNAL CABLE	25.	CIRCUIT BREAKER
7.	LOCKOUT SWITCH	26.	POWER CABLE (4-6 EA)
8.	PROCESS COMPUTER	27.	AC POWER DISTRIBU-TION PANEL
9.	SIGNAL CABLE		
10.	KEYBOARD	28.	CABLE (4 EA)
11.	SCREW (4 EA)	29.	SCREW (4 EA)
12.	ACCESS PANEL	30.	COVER
13.	POWER SWITCH	31.	TERMINAL STRIP
14.	POWER CABLE (2 EA)	32.	SCREW (4 EA)
15.	SIGNAL CABLE (3 EA)	33.	SCREW (6 EA)
16.	COAXIAL CABLE	34.	DYNAMIC MIRRORS DRIVER
17.	POWER SWITCH		
18.	LASER POWER SUPPLY	35.	CABLE (4 EA)
19.	CABLE (3 EA)	36.	POWER SWITCH



746-02-01(1).

Figure 2-2. Electronic Cabinet Components (Sheet 1 of 2)

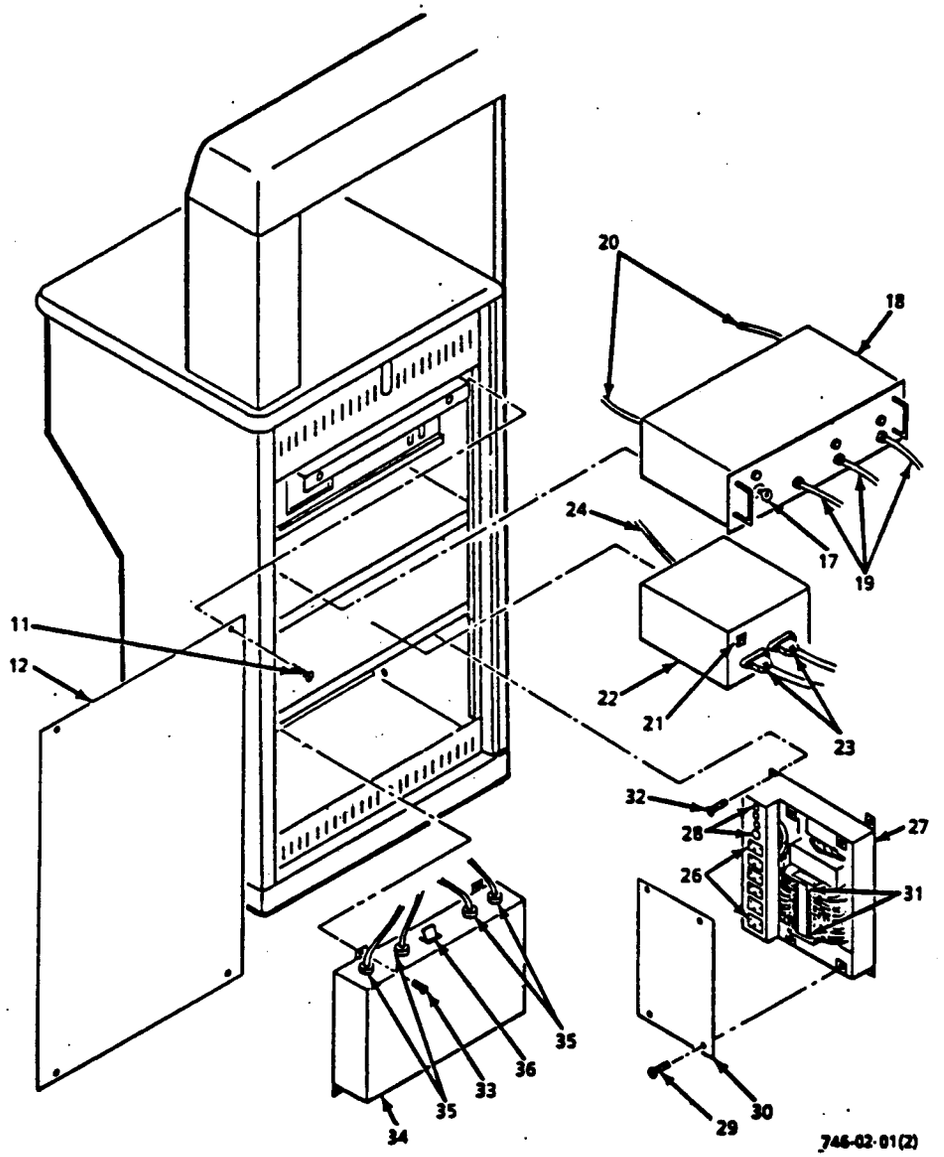


Figure 2-2. Electronic Cabinet Components (Sheet 2 of 2)

c. Disconnect power cable (5) from monitor.

d. Use a small flat blade screwdriver to remove two connector screws and disconnect signal cable (6) from monitor.

e. Remove monitor.

f. Install new monitor.

g. Connect signal cable (6) to monitor and use a small flat blade screwdriver to install two connector screws.

h. Connect power cable (5) to monitor.

i. Set power switch (3) on rear of monitor to on.

j. Install cable access panel (2) and secure with two screws (1).

2.4.2 Keyboard.

a. Set keyboard lockout keylock switch (7) on process computer (8) to off.

b. Disconnect signal cable (9) from keyboard (10). The cable is terminated with a telephone jack.

c. Remove keyboard.

d. Install new keyboard.

e. Connect signal cable (9) to keyboard.

f. Set keyboard lockout keylock switch (7) on process computer (8) to on.

2.4.3 Process Computer.

a. Remove four screws (11) and remove access panel (12) from rear of electronic cabinet assembly.

b. Set keyboard lockout keylock switch (7) to off.

c. Set power switch (13) on rear panel to off.

d. Disconnect two power cables (14) (monitor power and ac input power) from rear panel.

e. Disconnect three signal cables (15) (keyboard, elevator driver, and dynamic mirrors driver) and one coaxial cable (16) (Ethernet) from rear panel.

f. Push process computer (8) out through front panel and remove.

g. Push new process computer through front panel.

h. Connect three signal cables (15) and one coaxial cable (16) to rear panel.

i. Connect two power cables (14) to rear panel.

j. Set power switch (12) on rear panel to on.

k. Set keyboard lockout keylock switch (7) to on.

l. Install access panel (12) on rear of cabinet and secure with two screws (11).

2.4.4 Laser Power Supply.

WARNING

IF THE HIGH VOLTAGE CONNECTORS ARE DISCONNECTED BEFORE POWER SWITCH IS TURNED OFF, HIGH-VOLTAGE ARCING WILL RESULT.

a. Remove four screws (11) and remove access panel (12) from rear of electronic cabinet assembly.

b. Set keylock POWER switch (17) on laser power supply (18) to off. Verify that POWER ON and two DISCHARGE ON indicators are extinguished.

c. Disconnect three cables (19) from unit.

d. Grasp laser power supply by two handles and slide out of cabinet just enough to gain access to two cables (20) on rear of unit.

e. Disconnect two cables (20) from rear of unit.

f. Remove laser power supply from cabinet.

g. Install new laser power supply (18) into cabinet just far enough to connect cables.

h. Connect two cables (20) to rear of unit.

i. Push laser power supply completely into cabinet.

j. Connect three cables (19) to unit.

k. Set keylock POWER switch (17) to on. Verify that POWER ON and two DISCHARGE ON indicators are illuminated.

l. Install access panel (12) on rear of cabinet and secure with four screws (11).

2.4.5 Elevator Driver.

a. Remove four screws (11) and remove access panel (12) from rear of electronic cabinet assembly.

b. Set power switch (21) on elevator driver (22) to OFF.

c. Loosen two screws from each connector and disconnect two cables (23) from unit.

d. Slide elevator driver out of cabinet just enough to gain access to cable (24) on rear of unit.

e. Disconnect cable (24) from rear of unit.

f. Remove elevator driver from cabinet.

g. Install new elevator driver (22) into cabinet just enough to connect cable.

h. Connect cable (24) to rear of unit.

i. Push elevator driver completely into cabinet.

j. Connect two cables (23) to unit and secure with two screws at each connector.

k. Set power switch (21) to on.

l. Install access panel (12) on rear of cabinet and secure with four screws (11).

2.4.6 AC Power Distribution Panel.

a. Turn off main circuit breaker (25) on bottom rear of electronic cabinet assembly.

b. Remove four screws (11) and remove access panel (12) from rear of electronic cabinet assembly.

c. Disconnect power cables (26) from connectors J1 through J6 on ac power distribution panel (27). Four to six cables will be present.

d. Disconnect cables (28) from Molex connectors J7 through J10 on unit.

e. Remove four screws (29) and remove terminal board access cover (30).

f. Disconnect wires from terminal strip (31) nearest rear of cabinet. Wires are marked and must be disconnected from terminals 1, 2, 3, 6, 9, 10, 11, and 12.

g. Remove four screws (32) (two top and two bottom) and remove ac power distribution panel from cabinet.

h. Install new ac power distribution panel (27) in cabinet and secure with four screws (32).

i. Remove four screws (29) and remove terminal board access cover (30).

j. Connect wires to terminal strip (31) nearest rear of cabinet. Connect wires to terminals 1, 2, 3, 6, 9, 10, 11, and 12 as marked.

k. Install terminal board access cover (30) and secure with four screws (29).

l. Connect cables (28) to Molex connectors J7 through J10.

m. Connect power cables (26) to connectors J1 through J6 as required.

n. Install access panel (12) on rear of cabinet and secure with four screws (11).

o. Turn on main circuit breaker (25).

2.4.7 Dynamic Mirrors Driver.

a. Remove ac power distribution panel (paragraph 2.4.6, steps a through g).

b. Remove six screws (33) (two top and four bottom) from dynamic mirrors driver (34) and tilt unit down to gain access to cables.

c. Remove four cables (35).

d. Remove dynamic mirrors driver from cabinet.

e. Install new dynamic mirrors driver (34) in cabinet.

f. Tilt unit downward and install four cables (35).

g. Place unit flush against cabinet and install six screws (33) (two top and four bottom).

h. Set power switch (36) to on.

i. Install ac power distribution panel (paragraph 2.4.6, steps h through o).

2.5 OPTICS ASSEMBLY (Figure 2-3)

2.5.1 Laser.

a. Set keylock POWER switch (17) on laser power supply (18) to off. Verify that POWER ON and two DISCHARGE ON indicators are extinguished.

b. Turn two allen screws (1) ccw to release (one on each end) and remove laser cover (2).

c. Remove four screws (11, Fig. 2-2) and remove access panel (12) from rear of electronic cabinet assembly.

d. Disconnect three cables (19) from laser power supply.

e. Remove two screws (1) and remove cable access panel (2).

f. Pull three cables up through cable access hole.

g. Remove four cap screws (3, Fig. 2-3) from laser (4) (two front and two back).

h. Loosen allen screw (5) and remove interlock switch (6) from rear of laser.

i. Lift laser and remove from cabinet.

j. Install new laser (4) on cabinet and secure with four cap screws (3).

k. Install interlock switch (6) on rear of laser and tighten allen screw (5).

l. Push three cables (19, Fig. 2-2) through cable access hole to laser power supply (18).

m. Install cable access panel (2) on cabinet and secure with two screws (1).

systems, inc.

COMPANY CONFIDENTAL

LEGEND:

1. SCREW (2 EA)
2. LASER COVER
3. SCREW (4 EA)
4. LASER
5. SCREW
6. INTERLOCK SWITCH
7. SCREW (2 EA)
8. OPTICS COVER
9. SHUTTERS
10. SCREW (2 EA)
11. THUMBSCREW
12. BEAM-TURNING MIRROR
13. SCREW (4 EA)
14. BEAM EXPANDER
15. SCREW (2 EA)
16. DYNAMIC MIRRORS
17. SCREW (2 EA)
18. INTERLOCK SWITCH
19. SCREW (2 EA)
20. INTERLOCK SWITCH (2 EA)

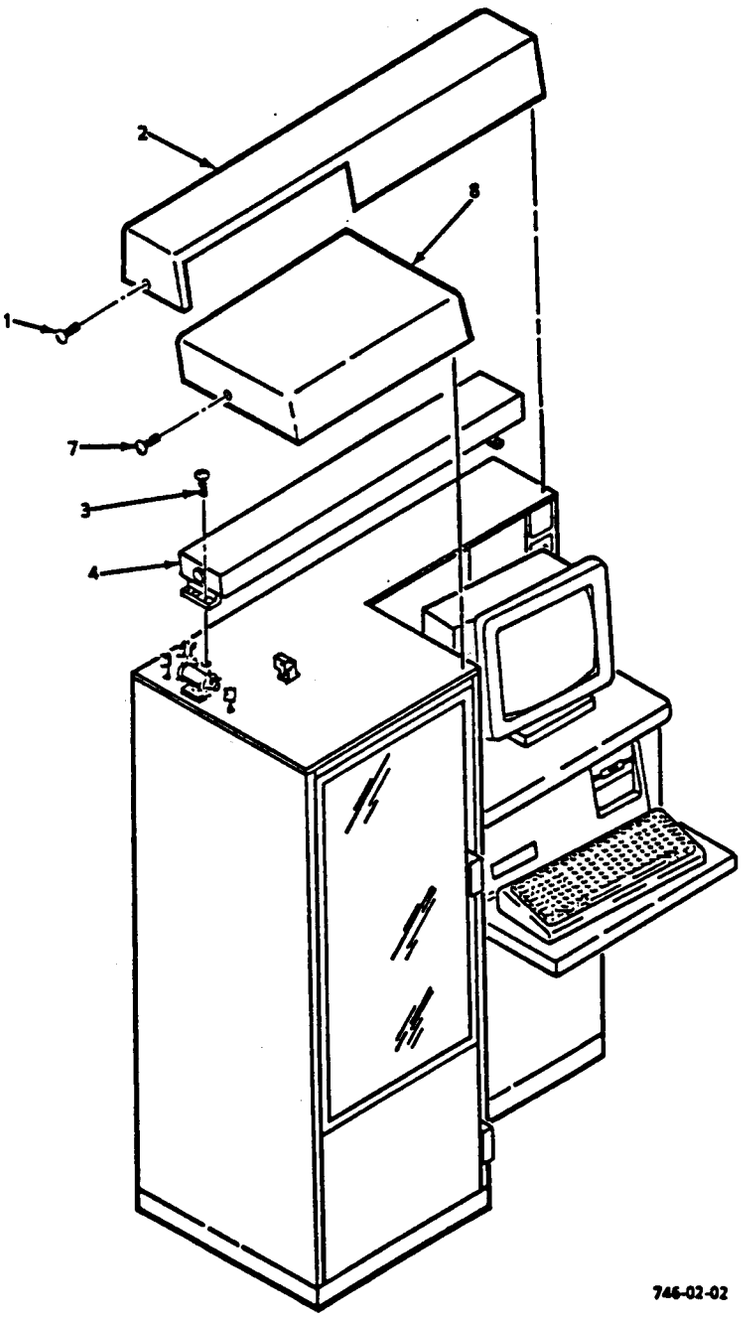


Figure 2-3. Optics Components (Sheet 1 of 2)

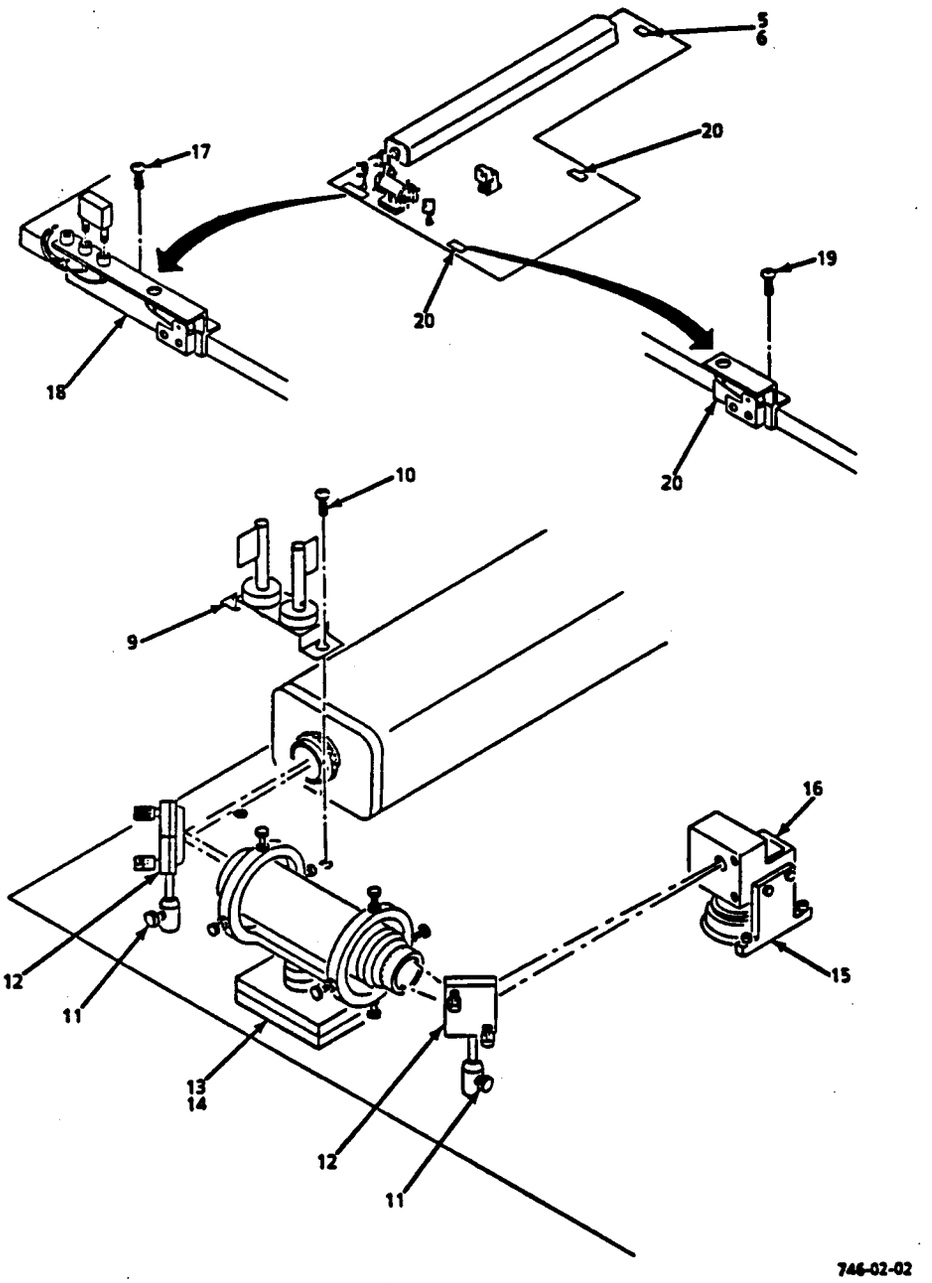


Figure 2-3. Optics Components (Sheet 2 of 2)

n. Connect three cables (19) to laser power supply.

o. Set keylock POWER switch (17) on laser power supply to on. Verify that POWER ON and two DISCHARGE ON indicators are illuminated.

p Install access panel (12) on rear of cabinet and secure with four screws (11).

q. Align laser mirrors (paragraph 3.3).

r. Install laser cover (2, Fig. 2-3) and turn two allen screws (1) cw to latch.

2.5.2 Shutters.

a. Turn two allen screws (7) ccw to release (one on each end) and remove optics cover (8).

c. Tag and unsolder wires from shutters (9).

d. Remove two allen screws (10) and remove shutters.

e. Solder wires to new shutters (9).

f. Install shutters and secure with two allen screws (10).

g. Install optics cover (8) and turn two allen screws (7) cw to latch.

2.5.3 Beam-Turning Mirrors.

a. Turn two allen screws (7) ccw to release (one on each end) and remove optics cover (8).

b. Loosen thumbscrew (11) and lift beam-turning mirror (12) straight up.

c. Install new beam-turning mirror (12) and tighten thumbscrew (11).

d. Align beam-turning mirrors (paragraph 3.4).

e. Install optics cover (8) and turn two allen screws (7) cw to latch.

2.5.4 Beam-Expander.

a. Turn two allen screws (7) ccw to release (one on each end) and remove optics cover (8).

b. Remove four allen screws (13) and lift beam expander (14) straight up.

c. Install new beam expander (14) and secure with four allen screws (13).

d. Install optics cover (8) and turn two allen screws (7) cw to latch.

2.5.5 Dynamic Mirrors.

a. Turn two allen screws (7) ccw to release (one at each end) and remove optics cover (8).

b. Remove four screws (11, Fig. 2-2) and remove access panel (12) from rear of electronic cabinet assembly.

c. Set power switch (35) on dynamic mirrors driver (33) to off.

d. Disconnect two outer cables (34) from dynamic mirrors driver.

e. Remove two screws (1) and remove cable access panel (2).

f. Pull two cables (34) up through cable access hole.

g. Remove two allen screws (15, Fig. 2-3) and lift dynamic mirrors (16) straight up from cabinet.

h. Install new dynamic mirrors (16) on cabinet and secure with two allen screws (15).

i. Push two cables (34, Fig. 2-2) through cable access hole to dynamic mirrors driver (33).

j. Install cable access panel (2) on cabinet and secure with two screws (1).

k. Connect two cables to dynamic mirrors driver.

l. Set power switch (35) on dynamic mirrors driver to on.

m. Install access panel (12) on rear of cabinet and secure with four screws (11).

n. Align beam-turning mirrors (paragraph 3.4).

o. Install optics cover (8, Fig. 2-3) and turn two allen screws (7) cw to latch.

2.5.6 Interlock Switches.

2.5.6.1 Laser Cover.

a. Turn two allen screws (1) ccw to release (one at each end) and remove laser cover (2).

b. Loosen one allen screw (5) and lift interlock switch (6) from rear of laser (4).

c. Tag and unsolder wires from switch.

d. Loosen two allen screws (17) and lift dual interlock switch (18) from cabinet.

e. Tag and unsolder wires from switch.

f. Solder wires to new interlock switch (6).

g. Install switch on rear of laser (4) and tighten one allen screw (5).

h. Solder wires to new dual interlock switch (18).

i. Install switch on cabinet and tighten two allen screws (17).

j. Install laser cover (2) and turn two allen screws (1) cw to latch.

2.5.6.2 Optics Cover.

a. Turn two allen screws (7) ccw to release (one at each end) and remove optics cover (8).

b. Loosen two allen screws (19) (one for each switch) and lift interlock switch (20) from cabinet.

c. Tag and unsolder wires from switches.

d. Solder wires to new interlock switches (20).

e. Install switches on cabinet and tighten two allen screws (19) (one for each switch).

f. Install optics cover (8) and turn two allen screws (7) cw to latch.

2.6 CHAMBER ASSEMBLY (Figure 2-4)

2.6.1 Heater/Fan.

a. Open chamber door (1).

b. Remove air filter (paragraph 2.6.1, steps a and b).

c. Set main circuit breaker on bottom rear of electronic cabinet assembly to off.

systems, inc.

COMPANY CONFIDENTAL

LEGEND:

1. CHAMBER DOOR
2. SCREW (2 EA)
3. BOTTOM COVER
4. HEATER/FAN
5. TERMINAL STRIP
6. SCREW (4 EA)
7. SIDE SHELF (2 EA)
8. KNOB (2 EA)
9. PLATFORM ROD (2 EA)
10. REACTION VAT
11. LID
12. LATCH (4 EA)
13. DRAIN VALVE
14. COLLECTION BOTTLE
15. SIDE HANDLE (2 EA)
16. SCREW (2 EA)
17. PLATFORM
18. CABLE
19. SCREW
20. CABLE CLAMP
21. SCREW (4 EA)
22. ELEVATOR
23. CHAMBER CABLE

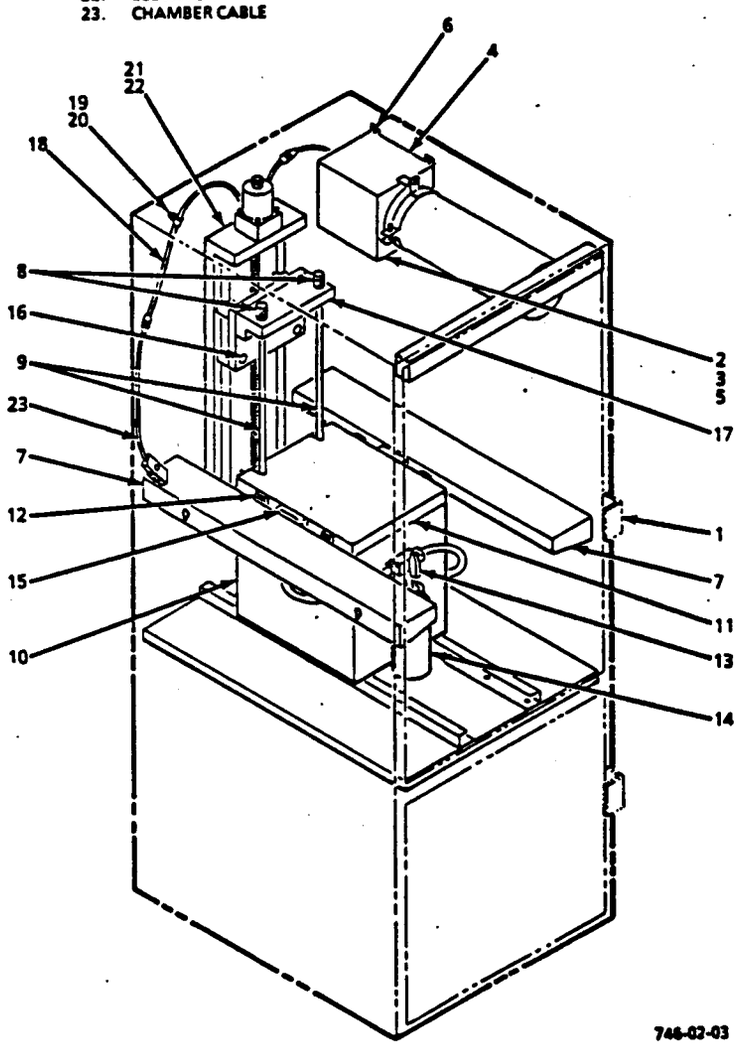


Figure 2-4. Chamber Components



systems, inc. COMPANY CONFIDENTAL

d. Remove four screws (2) and remove bottom cover (3) on heater/fan (4).

e. Disconnect three wires from terminal strip (5). The wires are marked.

f. Remove four screws (6) and remove heater/fan (4) from chamber.

h. Install new heater/fan (4) in chamber and secure with four screws (6).

g. Remove four screws (2) and remove bottom cover (3) from new heater/fan.

i. Connect three wires as marked to terminal strip (5).

j. Install bottom cover (3) on heater/fan and secure with four screws (2).

k. Set main circuit breaker on electronic cabinet assembly to on.

l. Install air filter (paragraph 2.6.1, steps c and d).

m. Close chamber door (1). Verify that fan is turning.

2.6.2 Reaction Vat.

a. Open chamber door (1).

b. Remove two side shelves (7) by lifting shelves to disengage from attaching screws.

c. Unscrew two knobs (8) to disengage platform rods (9) from vat (10).

d. Install lid (11) on vat and secure four latches (12).

e. Turn off drain valve (13) and remove collection bottle (14).

f. Grasp vat by two side handles (15) and slide out of chamber.

g. Clean resin from platform rods (9) (refer to User's Manual).

h. Slide new vat (10) into chamber.

i. Remove lid (11) from vat by releasing four snap fasteners.

j. Screw two knobs (8) to engage platform rods (9) to vat.

k. Place collection bottle (14) under valve (13) and turn on drain valve.

l. Fill vat with resin (refer to User's Manual).

m. Installl two side shelves (7).

n. Close chamber door (1).

2.5.3 Platform.

a. Open chamber door (1).

b. Remove two screws (16).

c. Unscrew two knobs (8) to disengage platform rods (9) from vat (10) and lift platform (17) straight out of chamber.

d. Install new platform (17) in chamber and screw two knobs (8) to engage platform rods (9) to vat (10).

e. Install two screws (16).

f. Close chamber door (2).

2.6.4 Elevator.

a. Open chamber door (1).

b. Disconnect cable (18) from left side of chamber.

c. Remove one screw (19) and remove cable clamp (20).

d. Remove platform (paragraph 2.6.3, steps a through c).

e. Remove four allen screws (21) and pull elevator (22) and cable straight out of chamber.

f. Install new elevator (22) in chamber and secure with four allen screws (21).

g. Install platform (paragraph 2.6.3, steps d through f).

h. Install cable clamp (20) on cable (18) and secure to cable and clamp to chamber with one screw (19).

i. Connect elevator cable (18) to chamber cable (23).

j. Close chamber door (1).

2.6.6 Beam Profiler. It is extremely difficult to remove the beam profiler in the Beta Site systems. Therefore, it is recommended that this unit not be removed. The SLA-1 can still operate even though the beam profiler is defective.

SECTION III
ALIGNMENT

3.1 INTRODUCTION

This section contains procedures to align the laser, optics, and chamber. These procedures must be performed when the optics or laser is replaced, or when these items become misaligned through normal wear.

3.2 TOOLS AND EQUIPMENT

Recommended alignment tools and equipment are listed in Table 3-1. Equivalent items may be used.

Table 3-1. Alignment Tools and Equipment

Nomenclature	Model/Part No.	Source
Ball driver, allen (2 ea)		Commercialy available
Power meter, laser		Liconix
T-square		Commercialy available
Caliper		Commercialy available
Tape, scotch		Commercialy available
Plumb bob		Commercialy available
Alignment tool, vertical		3D Systems
Level, precision		Commercialy available
Wrench set, allen		Commercialy available



systems, inc.

COMPANY CONFIDENTAL

3.3 LASER RESONATOR (Figure 3-1)

This is a very touchy alignment which should only be performed when laser power or beam shape has deteriorated. Two allen ball drivers (5/64 inch) and a laser power meter are required to perform this procedure.

a. Remove optics cover and laser cover.

b. Remove two screws and remove shutter assembly.

c. Place laser power meter to interrupt laser beam.

d. Insert one ball driver in upper rear laser adjustment hole and second ball driver in upper front laser adjustment hole.

e. Observe laser power meter and detune (turn 1/10 turn ccw) rear laser mirror adjustment.

f. Tune front laser mirror adjustment to maximize power. Adjust in small increments only.

g. Detune rear laser mirror (turn 1/10 turn cw) adjustment and repeat step f.

h. Repeat steps e and f several times until maximum power is obtained.

i. Place a card in front of the laser power meter and note shape of laser beam. The beam should be relatively round. If not, repeat steps e through h several times for a round beam shape at maximum power.

j. Carefully remove both ball drivers and remove card.

k. Insert one ball driver in lower rear laser adjustment hole and second ball driver in lower front laser adjustment hole.

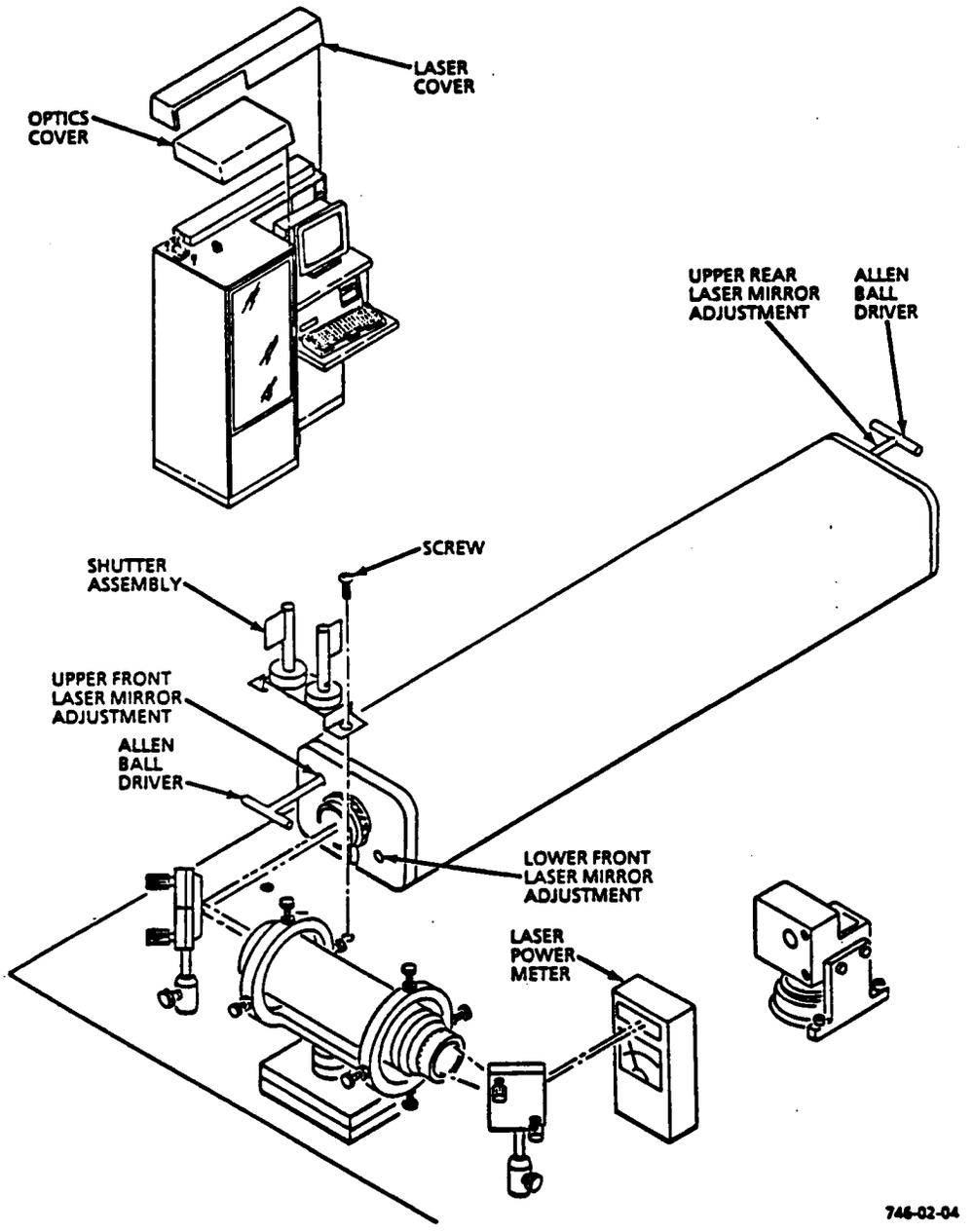


Figure 3-1. Laser Resonator Alignment

l. Repeat steps e through i several times until a round beam shape at maximum power is obtained.

m. Repeat upper and lower laser mirror adjustment several times to optimize laser power and beam shape.

n. Carefully remove both ball drivers and remove card.

o. Remove laser power meter.

p. Install shutter assembly and secure with two screws.

q. Install optics cover and laser cover.

3.4 OPTICS (Figure 3-2)

WARNING

UV LASER RADIATION MAY DAMAGE EYE TISSUE. AVOID DIRECT EXPOSURE TO LASER BEAM. WEAR SAFETY GLASSES.

CAUTION

Fingerprints may damage optics coatings. Clean hands thoroughly and strictly avoid touching optical surfaces.

a. Remove optics cover.

b. Loosen two adjustment screws (front and rear) and slide beam expander out and remove.

c. Adjust two each adjustment screws on two beam-turning mirrors and adjust mirrors so that laser beam strikes center of both mirrors and center window of dynamic window.

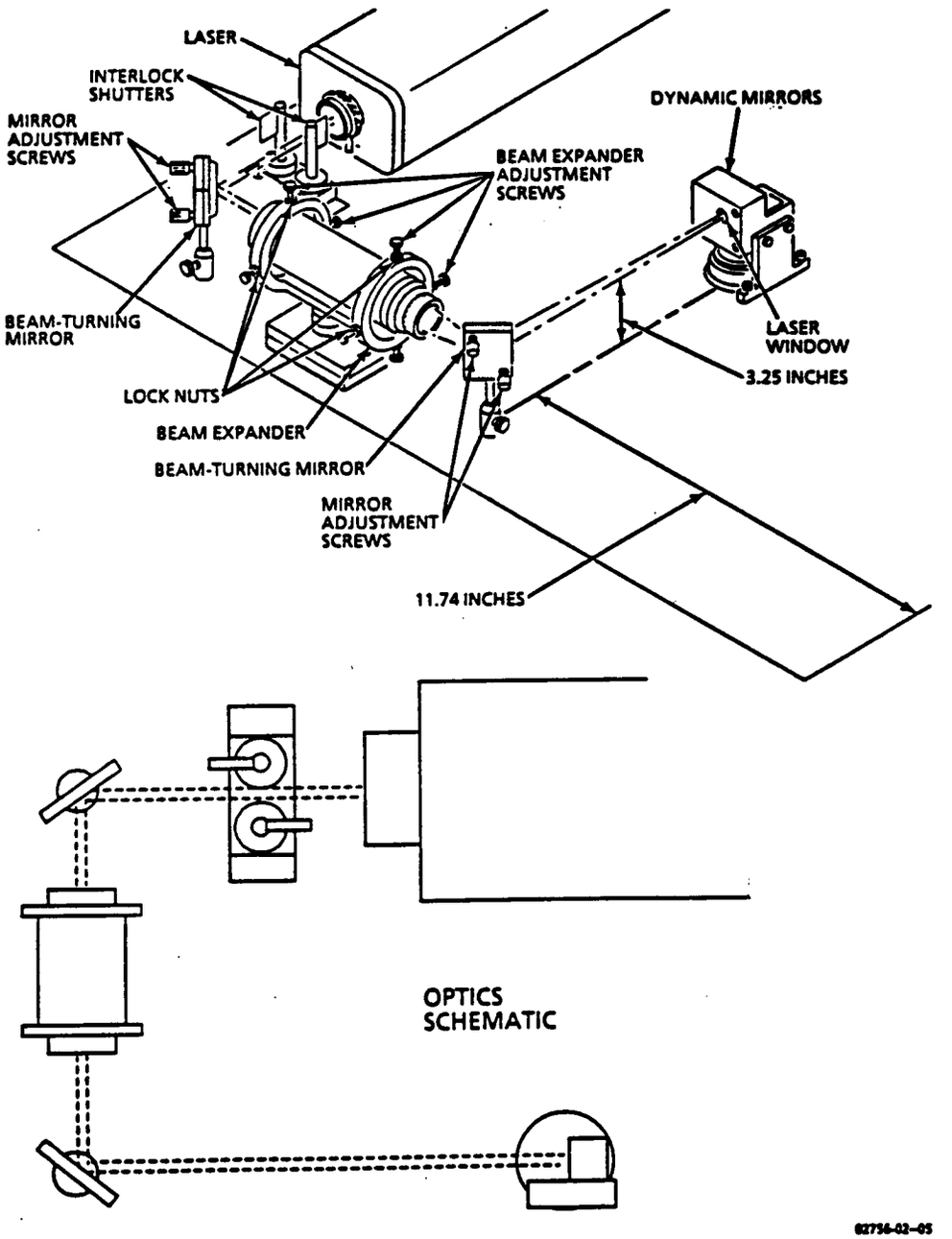


Figure 3-2. Optics Alignment

d. Use a T-square to check that beam path between second beam turning mirror and dynamic mirror is exactly 3.25 inch above optic plate along entire beam path. Use a set of calipers to take measurements.

e. Adjust two adjustment screws on second beam-turning mirror to achieve correct value.

f. Use a 12-inch set of calipers to check that beam path between second beam-turning mirror and dynamic mirror is exactly 11.74 inch from beam path to front edge of optic plate along entire beam path.

g. Adjust two adjustment screws on second beam-turning mirror to achieve correct value.

h. Repeat steps d through g several times for optimim vertical and horizontal alignment.

i. Put transparent scotch tape over input and exit apertures of beam expander. Make sure that tape does not contact lenses.

j. Install beam expander.

k. Adjust beam expander adjustment screws so that laser beam is centered as close as possible at input and exit apertures. Do not adjust any mirrors, only the beam expander. The position of the beam as it strikes the second beam-turning mirror and dynamic mirror should not change after installation of beam expander.

l. Tighten lock nuts on beam expender adjustment screws.

m. Remove scotch tape from beam expander apertures.

n. Install optics cover.

3.5 CHAMBER (Figure 3-3)

3.5.1 Vertical Alignment of Elevator. The perfect alignment of the elevator axis with the Vertical is critical for building of accurate stereolithographic parts. Any off-vertical alignment of components will result in a vertical skew of every part built with the SLA-1.

a. Remove reaction vat and platform (Section II).

b. On bottom of cabinet, raise up floor adjustment foot number 4 to balance cabinet on three feet.

c. Remove two allen screws and remove optic cover.

d. Open chamber door.

e. Remove two inside screws and lower optic window mount from inside chamber.

f. Suspend a plumb bob from center of optic window aperture.

g. Adjust height of string until top of plumb bob exactly touches floor of chamber.

h. Place a sheet of paper on bottom of chamber and mark plumb bob position exactly.

i. Install vertical alignment tool and place string against vertical string groove.

j. Run elevator up and down while observing plumb bob for lateral movement.

k. If plumb bob moves off from spot marked in step g, adjust chamber floor adjustment cabinet feet (1, 2, and 3) for best vertical alignment.

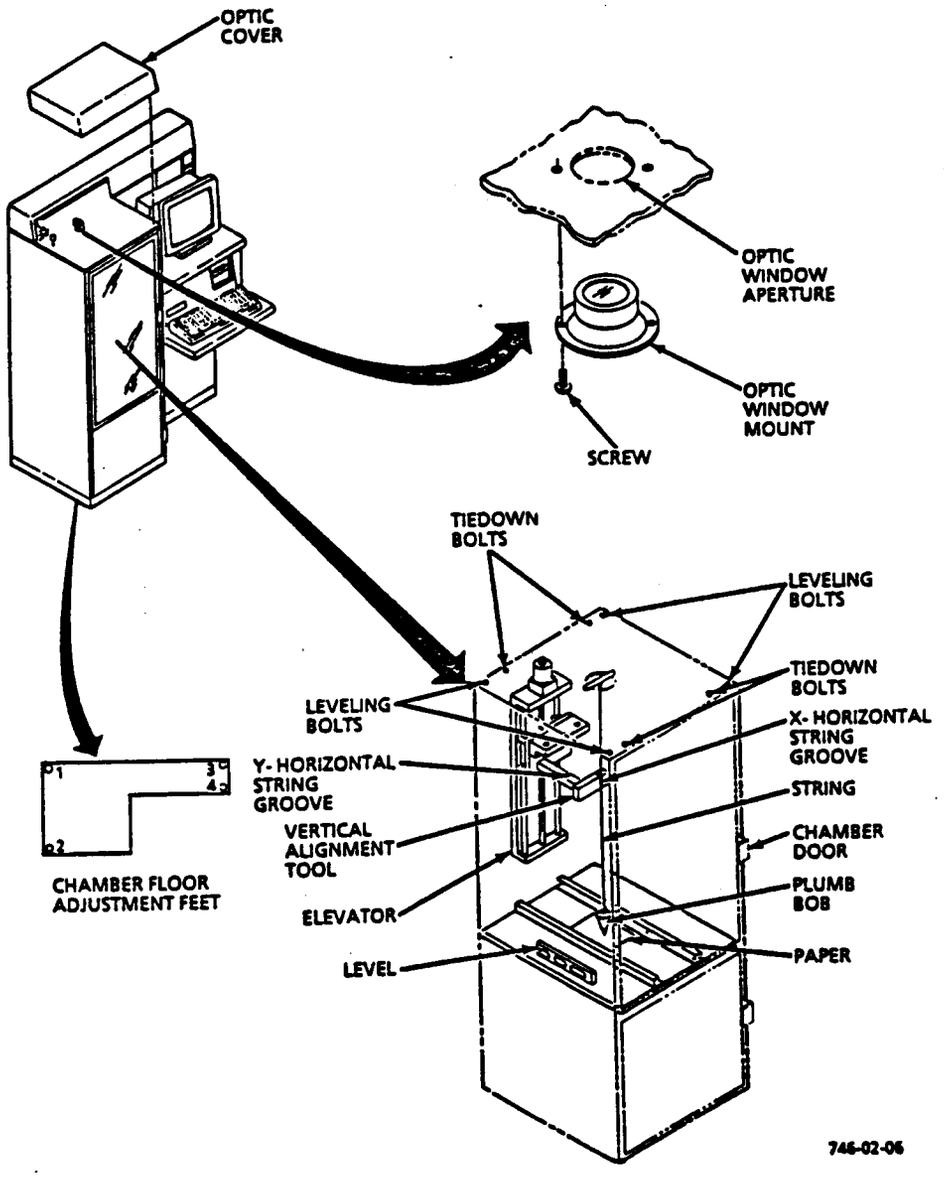


Figure 3-3. Chamber Alignment

l. Place string against horizontal string groove and mark plumb bob position.

m. Repeat steps j and k.

n. Repeat steps i through m several times until optimum leveling is achieved.

o. Lower foot number 4 to floor level. (Be careful not to upset leveling.)

p. Remove vertical alignment tool and plumb bob.

q. Install optic window mount by installing two screws inside chamber.

r. Close chamber door.

s. Install platform and reaction vat (Section II).

3.5.2 Horizontal Alignment of Optics Plate. Do not use the adjustable feet of the SLA-1 for horizontal alignment. This will upset the vertical alignment of the elevator.

a. Loosen four tiedown bolts on optic plate.

b. Place a precision level on optics plate.

c. Adjust four levelling bolts for optimum horizontal level at all points of the optics plate.

d. Tighten tiedown bolts.



systems, inc.

COMPANY CONFIDENTAL

SECTION IV
SPARE PARTS LISTS

4.1 INTRODUCTION

This section contains an exploded view illustration and listing of each authorized spare part.

4.2 HOW TO USE THIS SECTION

a. Locate the defective part or assembly in which the part is located in Figure 4-1. Note the index number.

NOTE

The SLA-1 is arranged into three major assemblies or component groups: the electronic cabinet assembly, the optics assembly, and the chamber assembly. These assemblies and components are illustrated and listed in Figure 4-1. Subassemblies and detail parts are listed in parts lists 4-2 through 4-4.

b. Locate the index number in the parts list for Figure 4-1 and note the part number, nomenclature and source for replaceable parts, or the figure number in which the replaceable parts are listed. Assemblies containing replaceable parts are listed for reference purposes only and are identified by the abbreviation REF in the Units Per Assy column.

c. Repeat steps a and b to locate subassemblies and detail parts in the referenced parts list.

d. Order replacement parts from the listed manufacturer in accordance with the instructions in paragraph 4.3.



systems, inc.

COMPANY CONFIDENTAL

4.3 MANUFACTURER'S NAMES AND ADDRESSES

SLA component manufacturers are listed in Table 4-1. When ordering spare parts, specify the part number, nomenclature, and quantity required. Always inform your 3-D Systems project engineer first if problems arise. He will usually be able to arrange to acquire the necessary parts. The manufacturers of SLA-1 components and subassemblies are listed for convenience.

Table 4-1. Vendor Names and Addresses

Manufacturer	Address and Telephone
3D Systems, Inc.	12847 Arroyo St. Sylmar, CA 91342 (818) 898-1533 FAX (818) 361-5484
General Scanning, Inc.	500 Arsenal St. Watertown, MA 02272 (617) 924-1010
Liconix	1390 Borregas Ave. Sunnyvale, CA 94089 (408) 734-4331 TWX 910-379-6475
Newport Corp.	18235 Mount Baldy Circle Fountain Valley, CA 92728-8020 (714) 963-9811
Wyse Technology	3571 North First St. San Jose, CA 95134 (408) 433-1026

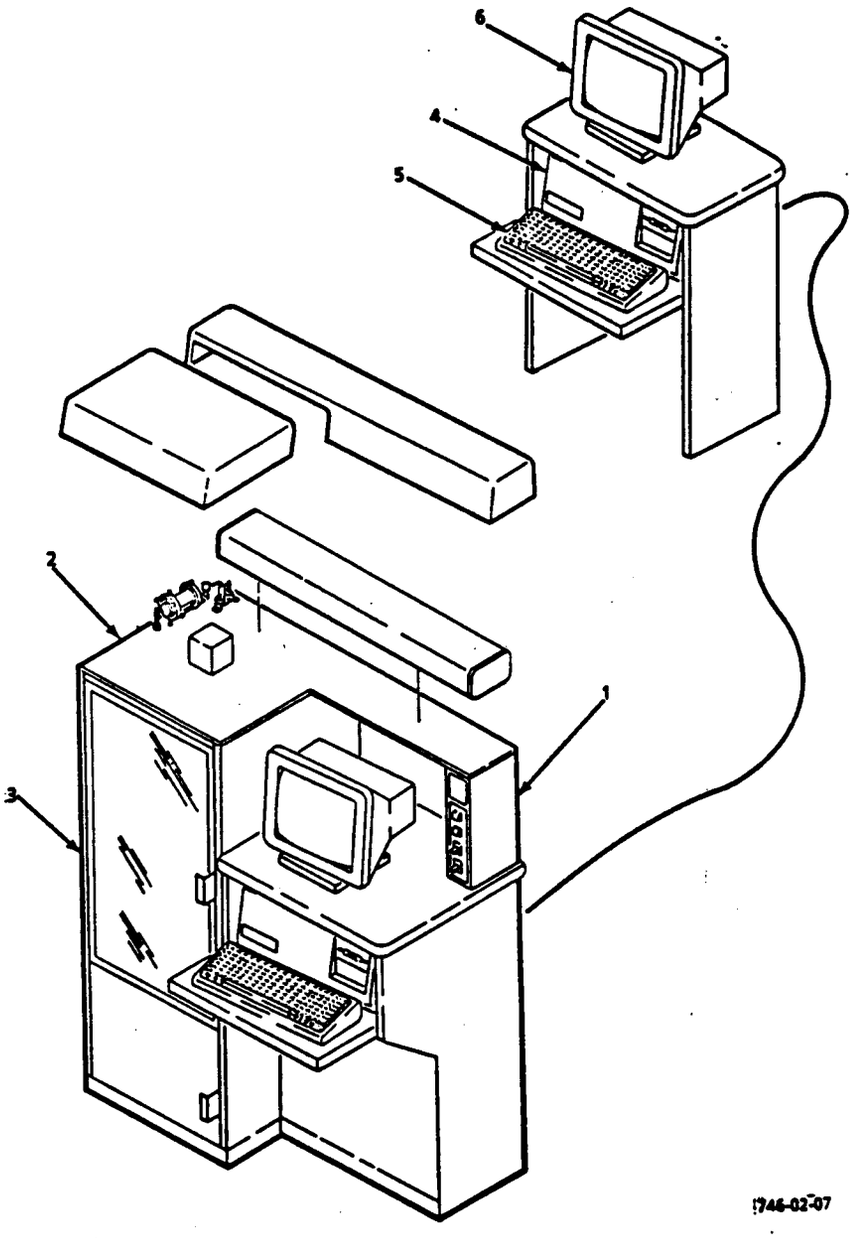


Figure 4-1. SLA-1 Stereolithography System

systems, inc.

COMPANY CONFIDENTAL

Figure & Index Number	Part Number	Description 1 2 3	Units Per Assy	Manufacturer
4-1		SLA-1 STEREOLITHOGRAPHY SYSTEM, Beta Site		
1	16135	• CABINET ASSY, Electronic (See Figure 4-2 for detail parts)	1	3D Systems
2	4240NB	• OPTICS ASSY (See Figure 4-3 for detail parts)	1	3D Systems
3		• CHAMBER ASSY (See Figure 4-4 for detail parts)	1	3D Systems
4	PC 386	• COMPUTER ASSY, Slice		Wyse
5		• KEYBOARD		Wyse
6		• MONITOR		Wyse

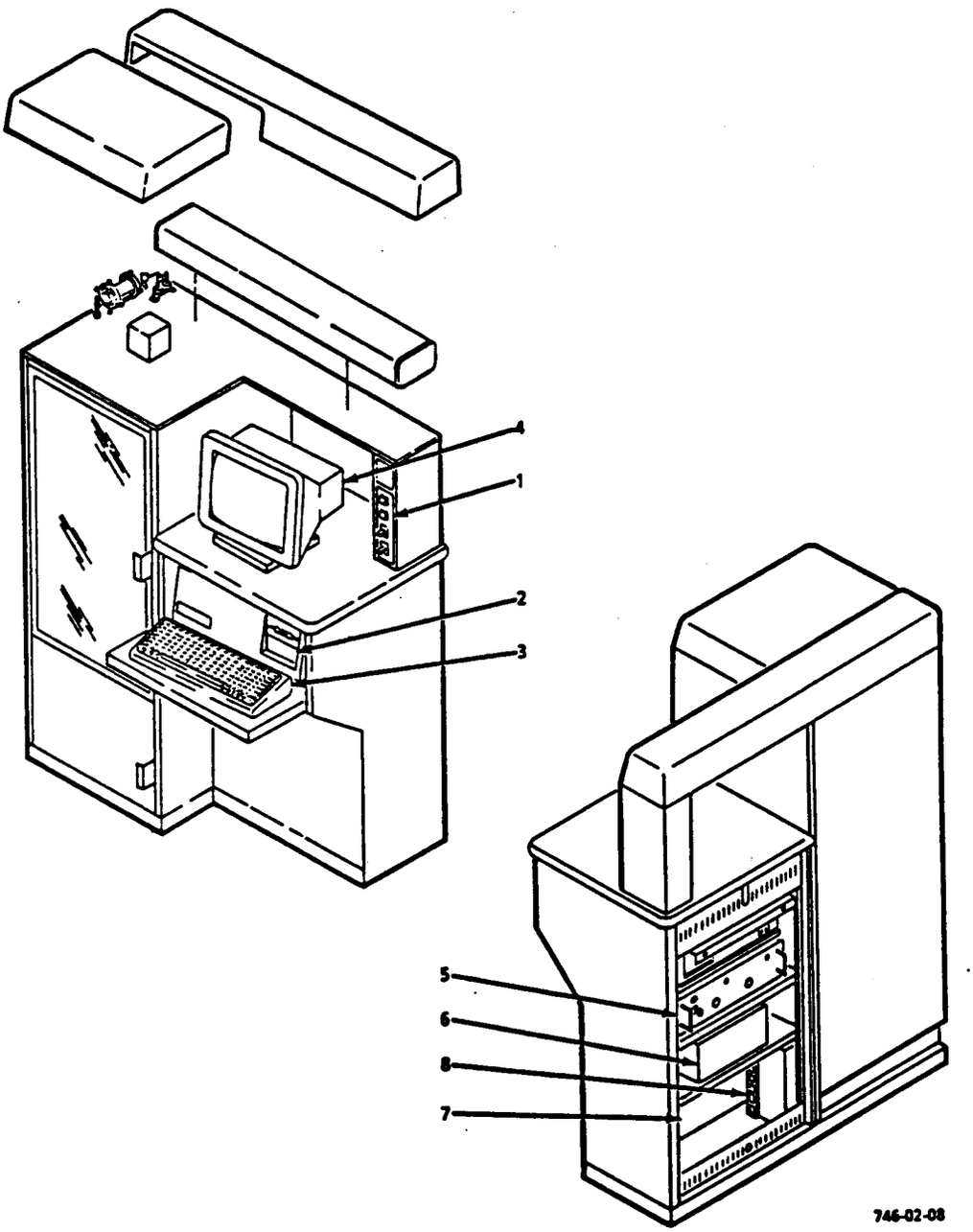


Figure 4-2. Electronic Cabinet Assembly



systems, inc.

COMPANY CONFIDENTAL

Figure & Index Number	Part Number	Description 1 2 3	Units Per Assy	Manufacturer
4-2		CABINET ASSY, Electronic (See Figure 4-1 for next higher assembly)	REF	
1	16180	. PANEL ASSY, Control	1	3D Systems
2	PC 286	. COMPUTER ASSY, Process	1	Wyse
3		. KEYBOARD	1	Wyse
4		. MONITOR	1	Wyse
5	16033	. POWER SUPPLY, Laser	1	3D Systems
6	16032	. DRIVER, Elevator	1	3D Systems
7		. DRIVER, Dynamic Mirrors	1	3D Systems
8	16130	. PANEL, AC Power Distribution	1	3D Systems

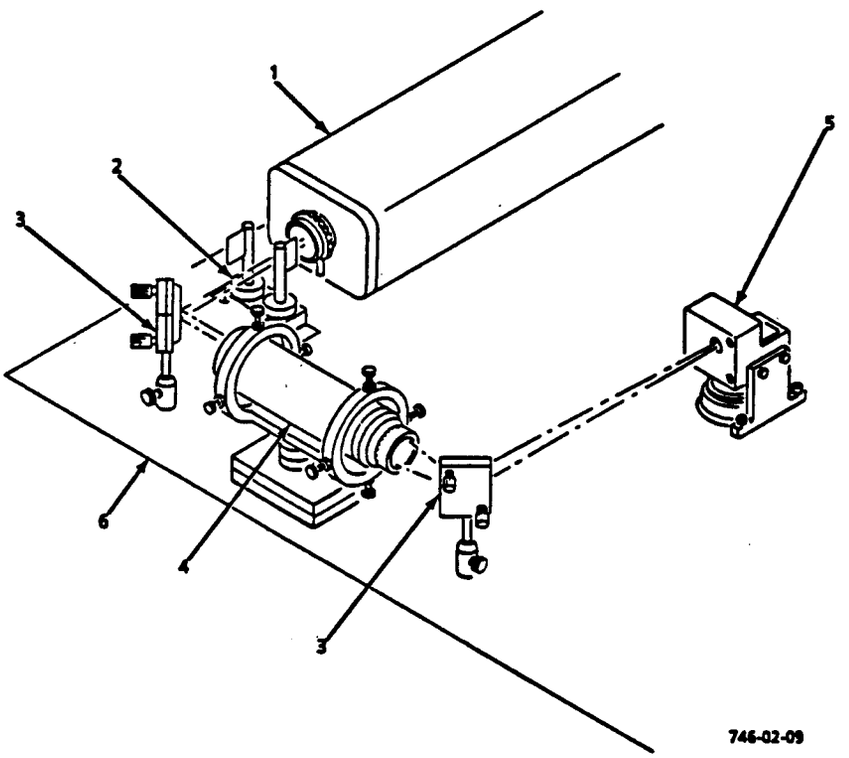


Figure 4-3. Optics Assembly



systems, inc.

COMPANY CONFIDENTAL

Figure & Index Number	Part Number	Description 1 2 3	Units Per Assy	Manufacturer
4-3		OPTIC ASSY (See Figure 4-1 for next higher assembly)	REF	
1		• LASER ASSY	1	Liconix
2	16160	• SHUTTER ASSY	1	3D Systems
3		• MIRROR ASSY, Beam Turning	2	Newport Corp.
4	D10493(4X)	• BEAM EXPANDER	1	3D Systems
5	Z2173	• MIRROR ASSY, Dynamic	1	General Scanning
6	16175	• PLATE ASSY, Optics	1	3D Systems

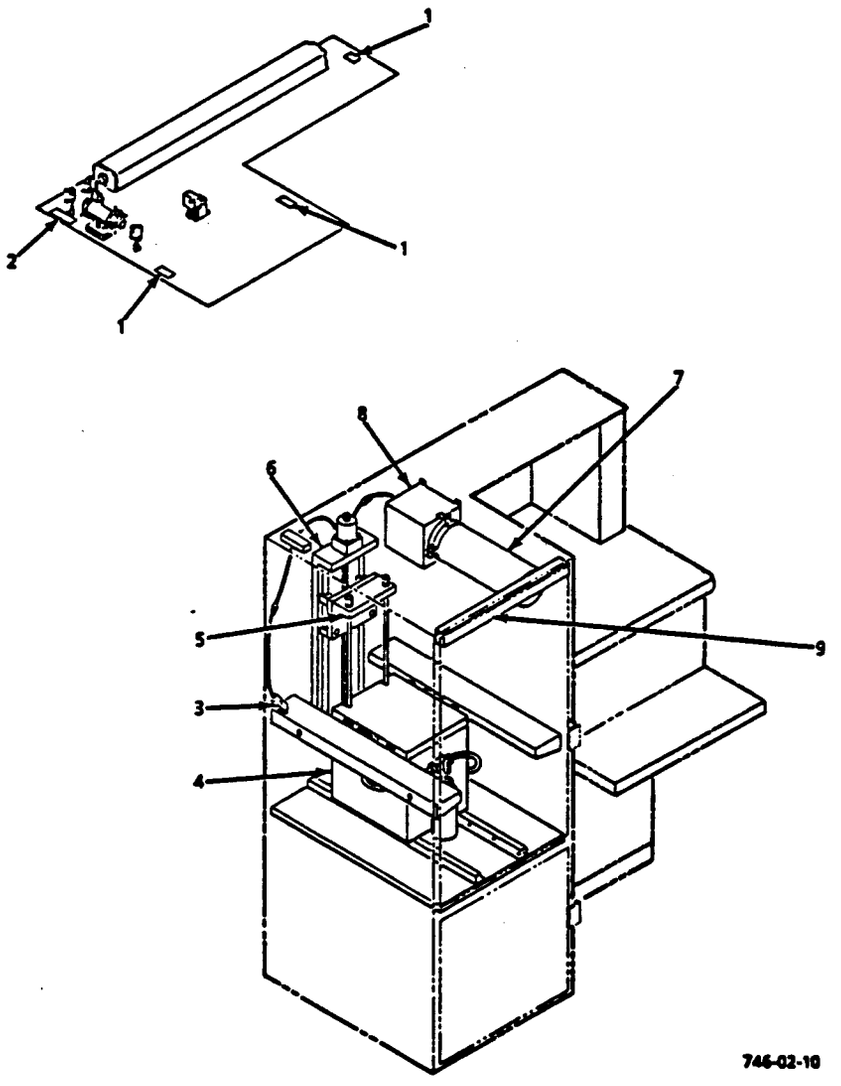


Figure 4-4. Chamber Assembly



systems, inc.

COMPANY CONFIDENTAL

Figure & Index Number	Part Number	Description 1 2 3	Units Per Assy	Manufacturer
4-4		CHAMBER ASSY (See Figure 4-1 for next higher assembly)	REF	
1	16175	. SWITCH ASSY, Interlock	3	3D Systems
2	16170	. DEFEAT ASSY, Interlock	1	3D Systems
3		. PROFILER ASSY, Beam	1	3D Systems
4	16115	. VAT ASSY	1	3D Systems
5		. ELEVATOR ASSY, Z-Stage	1	3D Systems
6		. PLATFORM ASSY	1	3D Systems
7	16125	. FILTER ASSY, Air	1	3D Systems
8		. HEATER/FAN ASSY	1	3D Systems
9		. LIGHT	1	3D Systems



systems, inc.

COMPANY CONFIDENTAL

SECTION V

WIRING DIAGRAM

5.1 INTRODUCTION

This section contains a wiring diagram to aid in troubleshooting and continuity tests.

5.2 WIRE LIST AND DIAGRAM

All SLA-1 cables and interconnections between components and subassemblies are shown in the wiring diagram (Figure 5-1).

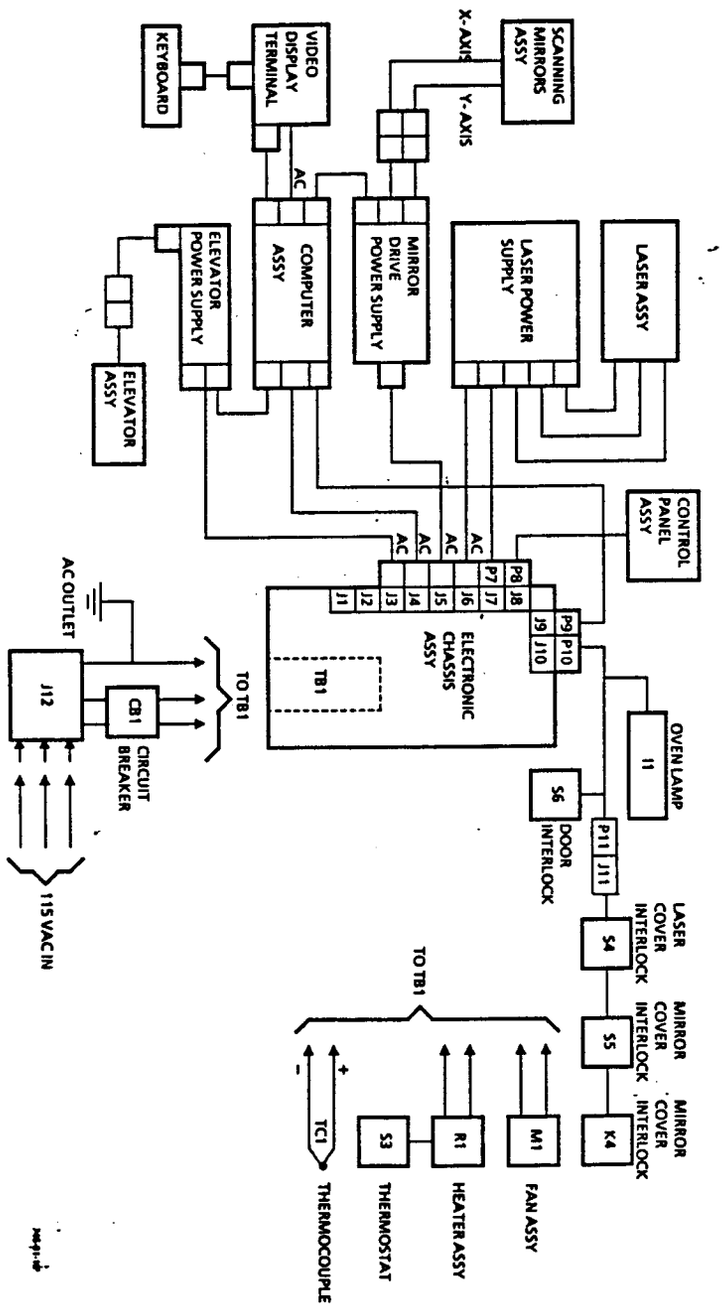


Figure 5-1. SLA-1 Wiring Diagram

5-2

systems, inc.
COMPANY CONFIDENT.

APPENDIX B

© 1989 3D SYSTEMS, INC.
ALL RIGHTS RESERVED

3D Systems, Inc.

SLA-1 Software Manual

Beta Release, First Draft

October. 1987

3D Systems, Inc.
12847 Arroyo Street
Sylmar, CA 91342
(818) 898 - 1533
(818) 361 - 5484 FAX

3D Systems SLA-1 Software Manual
Beta Release, First Draft
October, 1987

TABLE OF CONTENTS

1.0 Introduction 3
1.1 System Overview. 3
1.2 Major Part-Making Components 5
1.3 Utilities. 7

2.0 Getting Started. 8
2.1 Getting Around on the Slice Computer 8
2.1.1 Making a User Account. 8
2.1.2 Disk Usage Maintenance 9
2.2 Getting Around on the Process Computer 10
2.2.1 Disk Usage Maintenance 10

3.0 SLICE and the User Interface 12
3.1 The TELNET Communications Program. 12
3.2 SLICE Overview 13
3.3 SLICE Features and Capabilities. 16
3.4 SLICE Command Line Parameters. 18
3.5 The SLICE User Interface 20
3.5.1 UI Overview. 20
3.5.2 UI Main Command Menu 21
3.5.3 UI Alter Options Menu. 22
3.5.4 UI Extra Parameters Screen 24

4.0 The Parser 25
4.1 The FTP File-Transfer Program. 25
4.2 Operating The Parser 25
4.3 The Parser's Options 27
4.4 Critical Areas 28

5.0 The Supervisor 29
5.1 The Steps To Take Before Running the Supervisor. 29
5.1.1 Getting the Step Periods 29
5.1.2 Editing the Supervisor's Default Parameters File . . .30
5.1.3 Editing the Layer Control File 31
5.2 Operating the Supervisor 32
5.3 The Supervisor's Options 33
5.4 The Supervisor's Keyboard Commands 34
5.5 Using Rivets 35

6.0 Other Programs 37
6.1 FRONT, Simple Front End to Laser Controller. 37
6.2 CLIFF, Control Laser Instrument From File. 38
6.3 STEREO, The Stereolithography Device Driver. 40
6.3.1 STEREO Command Set 40
6.3.2 STEREO Options 42

7.0 File Formats 45
7.1 SLICE's Input File (*.STL) Format. 45
7.2 SLICE's Output File (*.SLI) Format 48
7.3 The Parser's Output File (*.P and *.L) Format. 49
7.4 Critical Area File (*.BOX) Format. 51
7.5 Supervisor Default Parameters File (SUPER.PRM) Format. .52
7.6 The STEREO.DEF Driver Default File Format. 54
7.7 The STEREO.GEO Geometric Correction File Format. 55

Appendix A - SLA-1 Software System Chart. 56

Appendix B - The STEREO Command Set 57

Appendix C - Stereolithography Software Glossary. 58

Appendix D - Editor Commands. 61

1.0 Introduction

Welcome to the world of Stereolithography. By now, you know what Stereolithography is all about, and you have seen an SLA-1 machine. The job of this manual is to introduce you to the software that makes a SLA machine work.

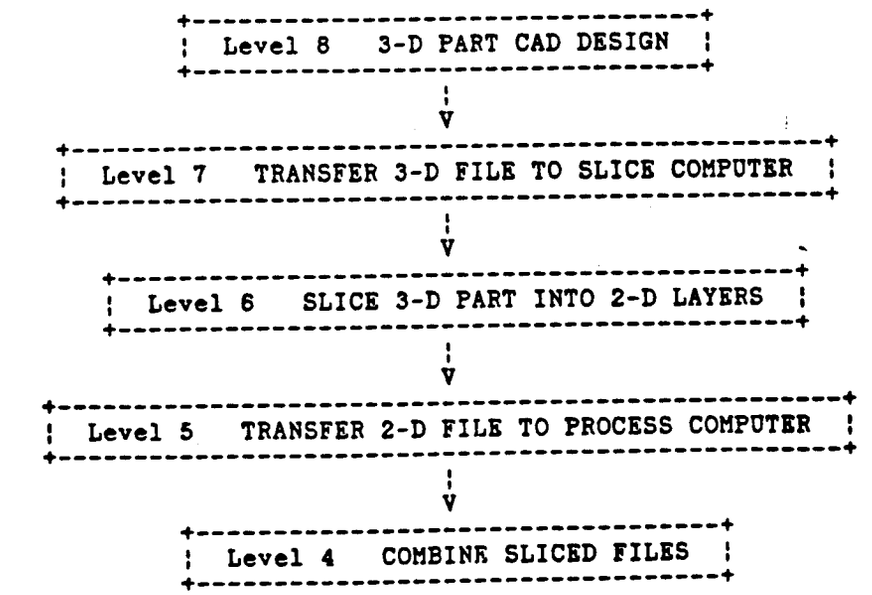
First, let's quickly review the Stereolithography process. A three-dimensional part is designed on a CAD system, and transferred to a Slice Computer where the part is transformed into a series of two-dimensional layers. From that computer, the two-dimensional part is moved over to the Process Computer in the SLA machine. It is the job of that computer to make the part.

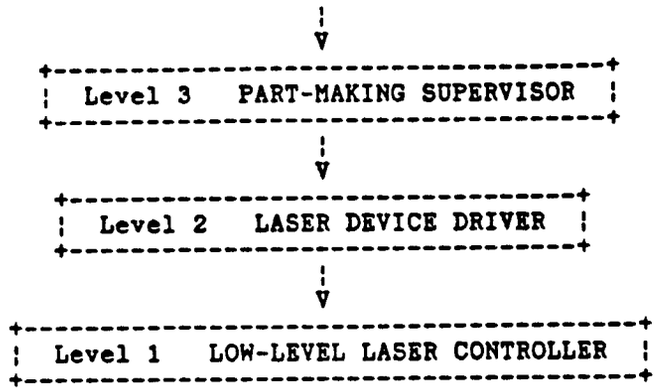
This manual follows the Stereolithography process. First, you will receive a brief introduction to all the important software programs. Then you will be shown how the Slice Computer works, how the Process Computer Works, and how to pass files to them.

1.1 System Overview

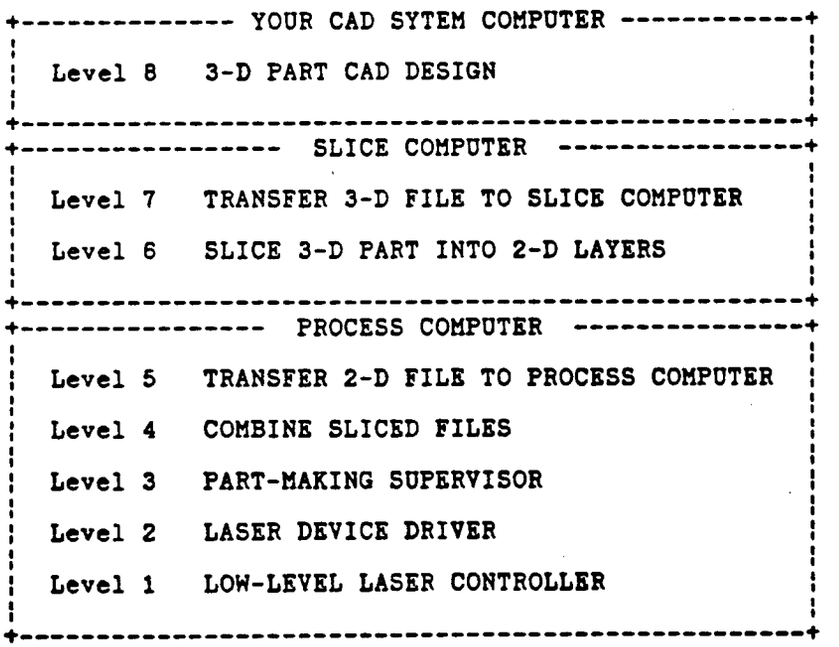
By System, we mean the Stereolithography Software System. And it is a system. It encompasses two operating systems - MSDOS and XENIX - and over a dozen programs, each important to some aspect of the Stereolithography process.

At last count, we figure there are nine levels of software. Each level can be viewed as a black box and having a unique task:





The responsibilities of providing these services are spread out amongst three computers -- one that you already use for CAD work, our 80386-based Slice Computer, and our 80286-based Part-Making Computer:



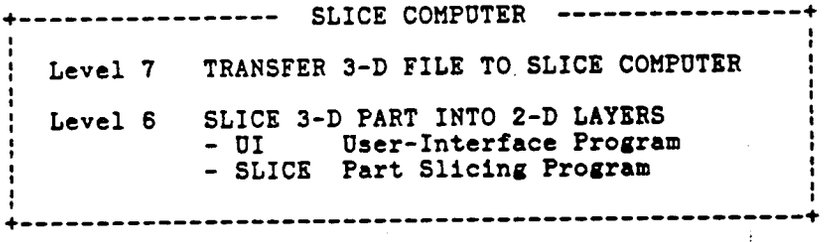
1.2 Major Part-Making Components

The Slice Computer

The 80386-based Slice Computer contains XENIX, a full-bore multi-tasking Operating System which allows many users to simultaneously sign-on, transfer CAD files, and slice them. On this computer you will be introduced to some system management programs and some Stereolithography software.

The Slice Computer contains the Slice Program (SLICE) and its convenient User Interface Program (UI). SLICE is the program that takes a 3-D database and transforms it into a 2-D one suitable for part-making. Since SLICE conforms to the Unix philosophy of how programs work (which by the way is not so intuitive), we have developed UI which is an convenient menu-driven front-end to SLICE.

You will use UI to input the parameters you want SLICE to see and then run SLICE through UI. Generally, unless you need to do some disk or system maintenance, you will not need to leave UI until you sign-off.



The Process Computer

The Process Computer has many programs. Rather than instruct you how to call up each program from DOS, we instead have developed a menu program which you can use instead. The menu program leads you to each of the major part-making programs, the utility programs, and provides some disk management functions.

The SLA machine operator will tap into the Slice Computer and pull out the 2-dimensional database of the part he next wishes to make. This is accomplished using Ethernet software programs TELNET and FTP. TELNET temporarily makes the Process Computer into a computer terminal so that you can sign-into the Slice computer and run the User Interface program. You can then use FTP to transfer the 2-D databases (okay, we'll start calling them sliced files) files over to the Process Computer.

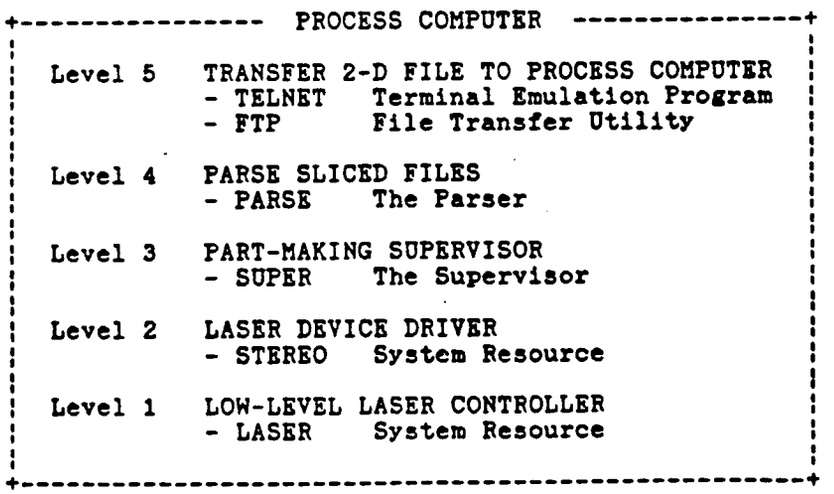
Once a sliced file is on the Process Computer, it may need to be combined with other sliced files, such as those containing a base for the part. The Parser, PARSE, does this job and also makes a special layer control file which you can edit to fine-tune the part-making process.

After running the Parser, you will proceed to the Supervisor, SUPER, which is the one program in charge of making a part. The Supervisor controls the motion of the laser beam and the Z-stage elevator, dipping and pausing for each layer.

Actually, The Supervisor's job is to "pass along" laser movement information to some rather sophisticated memory-resident software. The Supervisor talks to STEREO, a special Stereolithography Device Driver we created that knows about such SLA-specific things as special ways to draw and geometric correction which causes the laser beam to map linearly onto the vat's surface. STEREO, in turn, talks to LASER, the low-level laser controller, which is constantly running, moving the laser beam one discrete step every so often.

We told you a bit more than you probably needed to know, but we want you to get a good understanding of the system. The STEREO Device Driver has a set of commands that control how the laser moves on any given layer. Sometimes you want the laser to behave differently at some point during the part-making process. So, we gave the Supervisor the capability of passing along to STEREO commands that you have placed in certain files.

We want you to be aware that you are learning the commands of a common device driver, a driver that many different programs can use. Once you learn the STEREO Command Set, you will be in pretty good shape to quickly adapt to other programs.



1.3 Utilities

In addition to the major components, there are a variety of utility programs that aid the part-making process.

There is the power on-off sequencer (POWER) that enables you to safely turn on and off various elements of the SLA machine with appropriate directions.

And there is the Calibrator (CALIB) that is run not-so-often to calibrate the Vat Surface (the "Field") to make the SLA machine geometrically correct. Special hardware is necessary in order to use this program.

The Profiler (PROFILE) that analyzes the laser beam, and it is run often, perhaps once a day. It makes note of the current shape and intensity of the beam. The results of its analysis is retained in a special file that other programs can read.

The Get Step Periods program (GETSP) is one such program. It eliminates the need for you to be a chemist by reading the Profiler's data and enabling you to determine an appropriate Step Period (laser speed) for making a particular part. It knows the working curves for different materials, and it will select an appropriate Step Period for your forthcoming part-making sessions.

ZSTAGE gives you control over the Z-Stage, the vertical elevator that the part sits on. ZSTAGE is used to set the elevator just before a part is make, and it is also used to bring a part up out of the vat.

POWER	Power On/Off Sequencer
CALIB	Field Calibrator
PROFILE	Beam Profiler
GETSP	Determine Step Periods
ZSTAGE	Z-Stage Controller

Utility Programs

2.0 Getting Started

In this chapter, we assume you have installed your Slice and Process Computer Systems according the Installation Manual and have verified their proper operation. This chapter takes you at that point to help you get started using these two computer for the part-making process.

2.1 Getting Around on the Slice Computer

Each person performing slicing should have his or her own XENIX account on the Slice Computer. Once an account has been created, the owner can sign on.

2.1.1 Making a User Account

Sign onto the sysadm account by typing 'sysadm' at the login prompt and press Enter. Enter the password, which is '3dsys'. Note that the password isn't displayed as its typed in. And the case of the letters is important; type them in lower case.

The sysadm account is a special Xenix account that automatically takes you to a menu-driven system administration program. From this menu you can create and remove user accounts, and perform a varietry of other system tasks. If you want to know more information about the sysadm function than what is presented here, please refer to the appropriate Xenix documentation.

Take a few moments to look over the screen. Select the User Maintenance options by pressing 'U'. Do not press Enter.

Select MakeUser by pressing 'M'. Do not press Enter. Answer 'y' (and Enter) to 'detailed instructions?'. Next time you can answer 'n' because you will be familiar with the make-user process.

Enter the new user's login name. This should be short and in all lower case letters. We simply use people's first names and an occasional first letter of their last name. Make up your own rule, but be consistent. It will make system management easier.

Answer 'y' to 'Do you wish to use the next available user id?' query. Let the system assign the id number. You will rarely have to refer to it.

Answer 'y' to 'Do you want to use the default group? ' query. This will assign the standard group 'group'.

Assign a password that is at least 5 characters in length. You will have to enter it again as a safety precaution against typos. If you don't care about security, just make the password the same

for all users. This way, you can hop around the various user accounts and pick up the sliced file.

Select the Standard (Bourne) Shell, item 1. Enter a comment if you want. Generally, you will enter the user's full name (with upper and lower case letters) here.

Review the account infomation. If you are satisfied, answer 'n' to 'Do you want to change anything? ' prompt. The computer will take a few moments to create the user.

You have finished adding a user to the Slice Computer. That user can now sign-in and do slicing.

If you want to remove a user, you can select the Remove User option. You will have to delete all of the user's files before you can delete him from the computer system.

Play around with the sysadm menu system. Don't do anything that looks irreversible or serious until you have read up on it.

2.1.2 Disk Usage Maintenance

Depending on the size of your part files, you will need to take time to make periodic maintenance. This means you should erase or archive old part files to make way for new ones.

When you sign-in, the computer will tell you how much disk space is taken up by your files. The number shown is the number of 512 byte blocks (1/2 K). Check this number to make sure you aren't taking up more disk space than you thought. Make sure your disk usage doesn't get out of hand.

If you are in the User Interface, use option Q to quit and return to Xenix. There you can use the rm command to remove files:

```
rm sli0045.stl  will erase the single file sli0045.stl
rm gbox03*      will erase all files whose name begin with 'gbox03'
```

You can also use cp to copy and mv to move files around. Refer to the Xenix Documentation for more information.

If you feel you need more information about the Xenix Operating System, we recommend the following books:

* Understanding XENIX: A Conceptual Guide
  Que Corporation

* Working with XENIX Systems
  Scott, Foresman, and Company

* XENIX at Work
  Microsoft Press

* Inside XENIX
  The Waite Group

* XENIX Tutor
  XENIX Training on DOS PCs

SCO, our Xenix supplier sell these books to end-users. Contact them at (408) 425 - 7222.

2.2 Getting Around on the Process Computer

The Process Computer is your typical AT-compatible MSDOS Computer System. If you don't know the first thing about MSDOS, don't worry. All you have to do is turn the computer on; it does the rest. You will enter a Menu Program from which you can get to all the essential part-making programs.

2.2.1 Disk Usage Maintenance

As with the Slice Computer, you will need to watch your disk usage on the Process Computer. The Process Computer contains 40 Megabytes, of which about 2 Megabytes are taken up by MSDOS and the Stereolithography Software.

Unless you have created your own directories for the sliced files, you will be in the \WORK directory. Periodically, go to the DOS Menu and select item 1 to get the disk usage stats. You will see something like:

```
 21344256 bytes total disk space
    49152 bytes in 2 hidden files
    45056 bytes in 22 directories
  6254592 bytes in 391 user files
    20480 bytes in bad sectors
 14974976 bytes available on disk

   655360 bytes total memory
   598128 bytes free
```

Watch the 'bytes available on disk'. This system has more than 14 Megabytes (14,000,000) left so there is no immediate danger of running our of disk space. Depending on the sizes of the part files, you should get concerned when this number drops below a few Megabytes.

Once you realize that you need to some housecleaning, or "diskcleaning" in this case, select item 3 in the DOS Menu. This will take you to a program called QuickDos II (QD2) where you can do such disk-related things as copying files, moving files, and deleting files.

To delete files, use the arrow keys to move up and down the directory listing shown. Press the Space Bar to tag a file. A white arrow should appear at the left of the file name. Keep

doing this until all the files you want to delete have been tagged. To untag a file, move back to the file and press the Space Bar again. The white arrow should disappear.

Press 'E' (no Enter) to erase. QuickDos will show you the file you have selected on a different screen. Press Alt-E (hold the Alt key down while pressing 'E') to begin the erase procedure. If you see something you don't like, press Esc to abort.

QD2 performs a variety of other useful disk management tasks. Refer to its documentation for more information.

3.0 SLICE and the User Interface

To use SLICE and the User Interface, you can sign-on to the Slice Computer either directly via its keyboard, or from the Process Computer over the Ethernet communications link. Section 3.1 describes how to use the Ethernet TELNET terminal program.

Once you are signed-in, there are two ways to tell SLICE what you want done. One is to work directly from the Xenix Operating System where you must specify all the options on one line. The other is to use UI, the Slice User Interface Program, where you can specify the options through a set of menus. Both techniwues are described below. We have found most people prefer using the User Interface.

3.1 The TELNET Communications Program

The TELNET Program allows you to connect up to the Slice Computer from the Process Computer and run the User Interface and SLICE programs.

From the Process Computer Main Menu, select item 1, Data Transfer. You will be placed in the XFER program are presented with the following options:

1. TELNET, Terminal Utility
2. FTP, File Transfer Program
Q. Quit

If you don't what to communicate just yet, press Q to quit and return to the main menu. Otherwise, select item 1, TELNET.

TELNET should run and connect the computer you are currently on (the Process Computer) up to the Slice Computer. If both computers are functioning and are properly linked together, you should see the Xenix Login message appear. At this point the Process Computer is like the Slice Computer's main console. You can now sign-in and run SLICE or the User Interface.

Eventually, you will have to sign-off. Get out of whatever program you may be in (the DEL key may do that) and type:

exit

You should then be logged off the Slice Computer and returned to the Process Computer's main menu.

3.2 SLICE Overview

This quick overview will give you a brief description of SLICE methodology.

SLICE's mission is to read in a solid model whose surfaces were tessellated into triangular facets, and then to calculate output that is suitable for the laser control system to polymerize the model in layers, working from the bottom up.

While we are discussing SLICE ,it is important to realize that these next questions touch upon deep topics. The areas of numerical analysis, analytical geometry, computer graphics, and computer architecture, are among those that are relevant to SLICE. The discussion below is by no means a thorough treatment of these topics, but rather a starting point to provoke thought.

Why use layers?

The UV catalyzed polymerization process usually takes place at or near the monomer-substrate surface, so an approach akin to assembling a loaf of bread by gluing the slices back together is taken by the laser process. SLICE "slices" the original model mathemtically which the laser then puts back together.

The laser must trace the outline of each slice and sweep across ("hatching") the interior of each slice. In addition, the end-slices of the bread loaf, and the near flat ones are traced more finely, called "filling" in SLICE parlance.

Therefore the object to be produced must be reduced to cross-sections. This is the heart of the problem--to do the cross-sectioning and hatching/filling efficiently and correctly.

Why use triangles?

There are many ways to represent solids, such as constructive-solid-geometry, parametric representation (based upon a variety of fundamental curves such as bicubic patches), and boundary representation (called b-rep in the literature). B-rep is the most popular and often the most efficient for a general class of objects (e.g. not wings or boat hulls).

The boundary representation method generally involves describing a solid model in terms of its surfaces which are approximated by polygons. The smaller the polygons the better the b-rep model represents the actual object. Some solid model analysis problems (such as tolerance checking) originate right here-because we stray away from reality once we take the first step of interpolating polygons. We can discuss this further in a future section.

How the model is actually produced, whether by volumes of revolution, extrusion, projection, cross-sectioning (in reverse of SLICE), etc., does not (at present) concern us here. But the end result is a model that must have integrity. Polygons must be planar, and adjacent polygons must indeed abut one another, for example.

The triangle is a very nice surface b-rep polygon to work with for several reasons: non-degenerate triangles always define a single plane in 3-space, (as two points define a line in 2-space, and 4 points define a hypercube in 4-space, etc.). A degenerate triangle is one in which one or more of the points have the same coordinates. Triangles are also the simplest entity with which to perform computations, as compared to trapezoids, hence they proliferate in many popular CAD and solid modelling systems.

Why have variable z-spacing?

Let us use our food example again. If we are slicing a loaf of bread that is geometrically uninteresting, say a loaf of Roman Meal, it is wasteful to slice at every millimeter; centimeters will do. If we are slicing a dome shaped French Bagette, for example, but want a reasonably accurate model, some slices should be thinner than others.

Those slices that have more detail and that have smaller slope should be smaller. This is a little like the resultion problem above, but now we will vary the grid size DYNAMICALLY as we move through the model. Secondly, this approximation corrects for limitations in the laser process, not for a mathematical loss of accuracy as in the integer problem.

The limitation in the laser process is caused by the fact that beam shoots at approximately a normal angle to the substrate surface. Angled surfaces are therefore built up in a stair-stepping method. The side of each step is a slice, and its angle is given by the angle of the laser beam relative to the substrate liquid surface.

The second limitation is that areas that are nearly flat must be treated as if they were completely flat. Completely flat regions are skin-filled, which is essentially a very fine cross-hatching. Near flat regions must be skin-filled lest gaps and/or warpage occur in the completed part. Additional structural features may have to be built in order to compensate for properties inherent in the polymer.

There are other effects relevant to the part-making process, but they are not covered here. There is for example, a correction necessary for the spherical projection of the laser beam onto the flat surface of the substrate liquid, which is done in a post-SLICE laser output phase.

The thickness of slices must vary then to: 1) obtain a sufficient

level of detail, and 2) provide sufficient polymerization structural strength in areas that have small slope.

Why are triangles classified?

Recall that the input consists only of triangles. Each slice will intersect some of the triangles.

Triangles are classified by their slope relative to the x-y plane in SLICE: flat (slope=0), near-flat (slope<human-determined subjective criteria), and "scan" triangles (slope is large).

The line segments that are calculated by slicing through triangles are treated differently depending on the triangle they came from. These resulting intersections of slices with triangles form outlines of slices, called "borders", and provide information needed to do the cross-hatching and skin-filling.

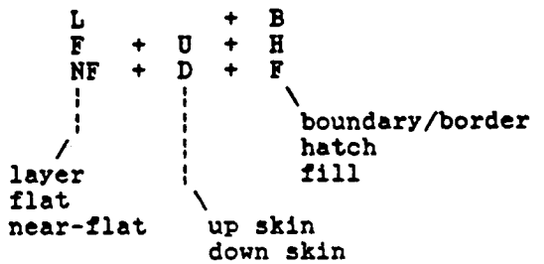
Why have output blocks?

Output blocks in Slice's output files identify the type of two-point vectors which follow. Special action can then be taken by the Supervisor on some of these blocks to strengthen the part and prevent warpage.

3.3 SLICE Blocks

SLICE takes the 3-D object database and classifies the facets as flat, near-flat, or scanned. When SLICE gets around to producing 2-D output, it organizes the layer vectors into various block.

SLICE block identifiers follow the following mnemonic rule. Not all combinations are valid.



New Block Mnemonic	Description
LB	Layer Border
LH	Layer Hatch
FUF	Flat Up skin Fill
NFUB	Near-Flat Up skin Border
NFUF	Near-Flat Up skin Fill
FDF	Flat Down skin Fill
NFDB	Near-Flat Down skin Border
NFDH	Near-Flat Down skin Hatch
NFDF	Near-Flat Down skin Fill

The above list is shown in kind of a triangle-type order which is different fromt the order SLICE outputs the data. SLICE outputs the way the part is to be built:

* borders first
* then hatches
* then down fills
* then up fills

SLICE outputs the blocks in the following order:

1.	L	Layer identifier, no vectors
2.	LB	Layer Border
3.	LH	Layer Hatch
4.	NFDB	Near-Flat Down skin Border
5.	NFDH	Near-Flat Down skin Hatch
6.	NFUB	Near-Flat Up skin Border
7.	FDF	Flat Down skin Fill
8.	NFDF	Near-Flat Down skin Fill

9.	NFUF	Near-Flat Up skin Fill
10.	FUF	Flat Up skin Fill

The L block isn't one really. It is followed by the interal Z-layer number, must like the old Z block. However, unlike the old Z which has the complete boundary for that layer, the L has no vectors.

3.4 SLICE Command Line Parameters

To activate SLICE the hard way, you will need to type out a XENIX command line with the form:

```
SLICE < input_file <options>
```

where SLICE is the current SLICE program (e.g. 'slice20b')
input_file is the name of the STL input file

<options> represent any combination of the following SLICE program options:

-scale <scale_value>
Used to specify a real number scale value that will be used to scale the input file coordinates. The default scale value is 1.

-sf <segment output file>
Used to specify an output file for z plane intersection segments. The default is to not generate a segment output file.

-zs <z spacing value>
Used to indicate the constant z spacing of the output cross-sections.

-hy <hatch spacing value>
Used to set spacing of hatch lines parallel to the Y-axis.

-hx <hatch spacing value>
Used to set spacing of hatch lines parallel to the X-axis.

-ha <hatch spacing value>
Used to set spacing of 60 degree angle hatch lines.

-hfy <hatch spacing value>
Sets spacing of hatch lines parallel to the Y-axis on flat surfaces.

-hfx <hatch spacing value>
Sets spacing of hatch lines parallel to the X-axis on flat surfaces.

-zsf <z spacing control file>
Used to specify the file containing variable Z-spacing information.

-c Centering option.
Centers object coordinates around 32768, 32768.

-nsc No section codes option.
Suppresses section codes in the output file.

-msa Minimum surface angle option.
Indicates minimum surface angle for scanned facets. Legal values are between 0 and 90 degrees.

-mia Minimum intersect angle option.
Indicates minimum intersect angle for hatch lines. Legal values are between 0 and 90 degrees.

-b Binary input option.
Indicates that the input file is in binary format.

-d Display option.
Causes the program to display cross-sections on the screen as they are generated.

-h Help option.
Causes the program to print a help similar to this list of options and then stop.

-b Binary input option.
Indicates that the input file is in binary attributed format.

-aries ARIES attributed binary input option.
Use when binary file is ARIES and attributed.

-disk Disk option.
Causes the program save intermediate information on disk rather than in memory.

-x or -y Slice Axis option.
Use the x-axis or y-axis as the slice axis, not the standard z-axis.

SLICE can be executed directly as a standard Xenix program. However, we have a developed a special User-Interface program for SLICE, described below.

3.5 The SLICE User Interface

We have developed a special User Interface program to SLICE that runs under XENIX. It is a much easier way to run SLICE compared to typing command lines. UI, as the User Interface is called, can be run remotely through Ethernet.

UI, the SLICE User Interface, presents menus and allows the user to easily select and define the various SLICE options for a particular STL Slice Input File. The options for different STL files are remembered in special setup files with the same file name prefix, but with the extension ".UII" (UI Information).

UI also allows the user to define a special variable Z-spacing file (with the extension ".UIZ") that is used by SLICE to generate different-spaced layers for different sections of a part.

3.5.1 UI Overview

The complex nature of using SLICE especially on a Unix-based system prompted the development of the User Interface (UI) program. It is a simple easy-to-use front end for the Stereolithographer where, by selecting items off a menu, he can enter the Slice options for his latest part. A key feature of UI it that it supports variable Z-spacing, a difficult feature of SLICE to use outside of UI.

UI allows a SLICE user to change options and invoke the SLICE program. SLICE then slices the specified 3D object into 2D layers using the UI options.

UI can be invoked from any process Unix or Xenix account. It is assumed that such a system has already been created with the necessary software. It is recommended that each individual needing to use UI and SLICE be given their own account.

The command syntax for the UI command is:

```
ui <option_file_name>
```

where <option_file_name> is the name you wish to refer to the new set of options. If you wish, it can be related somehow to the part name. If <option_file_name> is omitted, the last option file name will be used and its option file loaded.

3.5.2 UI Main Command Menu

The User Interface's Main Command Menu is shown below:

```
-----------------------------------------------------------------
                    - SLICE USER INTERFACE -
                         Version 2.10


    Current Object:  test
Database File Name:  test.stl
  Extra Parameters:  -aries -x

 Alter   - Alter standard options
 Extra   - Set extra parameters

 Load    - Load a different object's options
 Save    - Save current object's options
 Write   - Save current option's under different object name
 Copy    - Copy another object's options into current object

 DoSlice - Do slicing with current options
 Quit    - Exit UI (Use SAVE to save options first)

Which Command? _
-----------------------------------------------------------------
```

The UI commands behave as follows:

Alter - Go to the Options Menu so that the Slice Options can be changed.

Extra - Go to the Extra Parameters Screen so that non-standard Slice options can be specified.

Load - Loads a different set of options into UI. The previous objects options (if any) are not automatically saved. Using Load is equivalent to quitting UI and rerunning it with the new object's name.

Save - Saves the current option data into the current object file. Stays in the main menu.

Write - Same as Save, but allows you to specify a different file in which to store the option data. The new name becomes your current object file name.

Copy - Copies the options of a different object into the current object's options. The Copy command saves having to reenter options where options for a new object are being defined but the options are similar to those for some past object. By Copying those old options in and making

minor adjustments, you are off and running SLICE quickly.

DoSlice - Go Do Slicing with the current options. DoSlice allows you to specify what version of Slice you wish to use.

Quit - quits UI. Does not Save. Returns to back to the Unix or Xenix command level.

3.5.3 UI Alter Options Menu

All the numbered options in the UI program are parameters to the SLICE program. All the lettered options are used for UI control. Most of the SLICE options are self-explanatory.

```
------------------------------------------------------------
                    - SLICE USER INTERFACE -
                         Version 2.10

    Current Object: test

Option  Description                                  Value
------  -----------                                  -----
 (1)    DATABASE File Name                           test.stl
 (2)    Scale                                        1.000
 (3)    Z Spacing                                    10.000 Fixed
 (4)    X hatch spacing                              0
 (5)    Y hatch spacing                              0
 (6)    60/120 degree hatch spacing                  0
 (7)    X Skin fill for near flat surfaces           0
 (8)    Y Skin fill for near flat surfaces           0
 (9)    Minimum Surface Angle for scanned facets     0
 (10)   Minimum Hatch Intersect Angle                0
 (11)   Segment Output File Name                     test.sli
 (12)   Centering                                    Off
 (Q)    Quit, return to Main Menu

Alter which Option? _
------------------------------------------------------------
```

(1) DATABASE File name sets the name of the file that contains the three-dimensional CAD database for the object. An extension of ".stl" is assumed.

(11) Segment Output File Name is the name of the file that SLICE will output to. An extension of ".sli" is assumed.

(3) Z Spacing is one of two types, Variable or Fixed. Fixed is where just one value for all spacing, while Variable allows many. When Fixed is selected, UI just asks for the one Z spacing value. When Variable is selected, UI displays a new screen and allows you to edit the Variable Z-Spacing Table.

While adding and deleting values to and from the variable Z-spacing table, you will use the following commands:

A level setting	Add new level and spacing value to table
D level	Delete level from table
S	Save table, return to main menu
Q	Quit, return to main menu, don't save table
H	print a Help message similar to this

A adds a new level and spacing value. Both of these numbers will be multiplied by scale as they are sent out to SLICE. In other words, they are in the original CAD units -- inches, feet, whatever. Also, the indicated spacing begins at the indicated level. The first entry goes from that level to just before the level indicated by the second entry, and so on. The new spacing and hence the new dip depth will be used after the vectors for the indicated layer are drawn.

D deletes a level. S saves the table and returns you back to UI's main menu. Q skips the save, just returning you to the main menu. Any entries you made will be lost. H prints out a convenient help message.

If after you create a Variable Z-spacing table, you go back to Fixed spacing, then return to Variable spacing, the original Z-Spacing Table is restored.

The Save command in the Variable Z-spacing section does not save the table to disk. You must also use the Save or Write command in the main menu to make any changes permanent.

3.5.4 UI Extra Parameters Screen

Supplying Extra Parameters is a convenient way for us to support non-standard Slice commands without having to create confusing menus. You may never need to enter any extra parameters. This is a just-in-case feature.

Below is a copy of the Extra Parameters screen:

Extra Parameters allows you to specify non-standard slice options for inclusion in the slice command line. What parameters are valid is based on the version of Slice you will use.

The following parameters have been known to exist:

-x	part's height is in the x-axis
-y	part's height is in the y-axis
-aries	use the non-standard Aries CAD System input file format

Current Extra Parameters are:
-aries -x

Enter new Extra Parameters or just press Enter to keep old:

4.0 The Parser

The Parser takes sliced files and prepares them for the Supervisor. PARSE's name comes from the fact that it parses out the slice block identifiers and stores them is a special layer control file. This file you can later edit to fine tune the part-making process.

PARSE also combines multiple slice files together, such as a part and an appropriate base, and generates just one large vector file for the Supervisor.

4.1 The FTP File-Transfer Program

You can use the FTP File Transfer Program to move a sliced file over from the Slice Computer to the Process Computer. Go to the XFER program by selecting item 1 in the Main Menu. There, select item 2, for FTP.

FTP will prompt you for the Remote User Name. Type in the name of your user account. Then it will prompt you for your password. Type than it. If FTP can sign onto the Slice Computer, i will display the prompt:

```
FTP>
```

At this point, there are a variety of commands you can use. We will focus on just five of them here -- mdir, ldir, get, put, and bye.

You can use 'mdir' to get a directory listing of the Slice Computer, and 'ldir' to get a directory listing of the Process Computer.

Use 'mdir' to see the contents of your Slice Computer directory. Is the file you want to send over there? If it is, make sure you are going to store it in the right place on the Process Computer by using 'ldir'.

If everything looks right, use 'get' to move over the file:

```
get <filename>
```

where <filename> is the name of the file you want to copy over. The original file on the Slice Computer will not be disturbed. Use 'ldir' again to make sure the file came over.

In a similar manner, you can use 'put' to copy a file from the Process Computer over to the Slice Computer.

That's it. Use 'bye' to end FTP. FTP will sign-on the Slice Computer and return you back to the Process Computer's main menu.

4.2 Operating The Parser

When PARSE is brought up, it display the following heading:

```
-----------------------------------------------
  PARSE, Slice Files Combine and Parse Program
  3D Systems Laser Stereolithography System
  Version 2.30  by 3D Systems  October 1987
-----------------------------------------------

  Slice File Names: _
```

At this point, type in all the names of the SLICE files (no more than 10 at a time), separated by spaces, to combine into a single part. The output files generated will be used by SUPER, the Supervisor, to build a part on the Stereolithography apparatus.

After pressing Enter, PARSE will prompt with:

```
  Output File Name: _
```

Here, just give the prefix; do not add any extensions (like *.DAT). PARSE will make two output files <part>.L and <part>.P, where <part> is the name you input here. <part>.L will be a short file contain layer information, and <part>.P will be the combined vector file of all the SLICE input files.

Then, PARSE will ask for three numbers:

```
  Slice scale factor? _
  Slice -zs layer spacing? _
  Layer thickness in mm (20 mil = 0.508mm)? _
```

In the near future, PARSE will obtain this information directly from SLICE, but for now, you have to repeat it. Please do not delight too much in being able to "fudge" the part by inputting different numbers from what SLICE was told as this capability will soon vanish. PARSE will pass this information along to SUPER.

PARSE then begins it run:

```
  Starting parsing ... Hit Ctrl-Break to abort.

  7800: L1 LB1 LH1 NFDB1 NFDH1 FDF1 NFDF1 L2 LB2
  7810: L1 LB1 LH1 L2 LB2 NFUB2 NFUF2 FUF2
    :    :    :
```

If you press Ctrl-Break at any point before PARSE has finished, you will abort the parse procedure. Incomplete <part>.L and <part>.P files will be left on your disk so that you can look at the intermediate results. If you don't want these files, delete them to conserve disk space.

The '7800', '7810' numbers shown above are references to the L blocks being read in from the SLI file. The number itself is from after the line in the SLI file that begins with an L. The letter-number combinations that follow show the block-type and input file number that PARSE is currently working on -- NFUB2 means that PARSE is working on the Near-Flat Up-Facing Skin Border for the indicated layer from the second input file.

When PARSE is done, it returns control back to Menu program, at which point you can either edit its files or run the Supervisor.

PARSE maintains increasing layer numbers. It will automatically resolve the condition where the input files may start and stop on different layers by juggling the input from the various input files. PARSE can handle up to 10 input files per run.

For a given input file name in the 'Slice File Names' prompt, one can add one or more of the following options:

/Z<offset>	-	offset Z-layer numbers
/N<set of ZXYISABJHVCFGKCD>	-	no, don't use blocks indicated
/O<set of ZXYISABJHVCFGKCD>	-	only use blocks indicated

/Z allows you to offset the Z-layer values for that input file. This is useful when you need to align two previously non-aligned CAD objects such as posts and the major part. /Z-100 will reduce the apparent Z-layer values for that file by 100. The actual physical shift is related to the SLICE scale factor.

/N and /O allow you to select which blocks to exclude or include, respectively. /N LH,NFDH will eliminate all section cross hatching, and /O LB,NFDB,NFUB will only allow the border blocks through. "N" stands for "No" and "O" stands for "Only". By the way, it isn't wise to use both /N and /O together since they are mutually exclusive. No matter what you type, PARSE will always include the L-block.

4.3 The Parser's Options

PARSE accepts a command line whose format mimics the input queries discussed above. This allows PARSE to be run from a *.BAT batch file without user intervention. The format is:

```
PARSE <file1> <file2> ... /OUTFILE<output file prefix>
  /SCALE<scale>/ZSPC<Z-spacing>/THICK<thickness>
```

/OUTFILE specifies the output file prefix, /SCALE and /ZSPC the SLICE scale and Z-layer spacing, and /THICK the layer thickness. You cannot use these options from the Menu program.

As indicated before, specifying the scale, Z-spacing, and the thickness is temporary and will soon be eliminated. If the Command Line is incomplete, PARSE will prompt for the missing information.

4.4 Critical Areas

PARSE supports critical areas, where the Supervisor can be instructed to perform or not to perform something within the confines of a box. For a given part with a file name prefix of <part>, PARSE attempts to read the <part>.BOX file. If it doesn't see the file, it simply prints out a warning message and continues. Otherwise, it reads in the file and analyzes it while parsing.

Two types of critical areas are supported: Cross-Hatching with Rivets (XV) and Ignore Cross-Hatching (XI). That is, all the cross-hatch vectors within the critical area confines will either become rivets (multi-pass draws for strength) or be ignored.

See Section 7.4 for the format of the *.BOX file. See Section 5.5 for a discussion on rivets.

5.0 The Supervisor

The Supervisor supervises the part-making process. It reads the vectors from a file, sending them to the laser controller, modifying them according to the layer control file.

The Supervisor runs for as long as it taskes to make a part. It sends the packets of vectors to the STEREO driver, and it controls the Z-Stage, moving it down and back up for a dip after each layer.

The Supervisor reads three files. SUPER.PRM contains the default laser controller parameters for each block. The <filename>.L part layer control file contains your override laser controller parameters. The <filename>.P part file contains all the vectors from all the files combined together by the Parser.

The Supervisor knows about each type of block and which number PARSE File it came from. For any block from any file, you can tell the Supervisor to alter certain laser controller parameters such as step size, and step period.

5.1 The Steps To Take Before Running the Supervisor

Run the Beam Profiler and the GetSP programs if necessary. Adjust the Z-Stage using the Z-Stage Controller program.

(More)

5.1.1 Getting the Step Periods

You should run the GETSP program each time you use a new material or after you have run the Beam Profiler. GETSP will ask you for the desired lineheight (in mils) and make an estimate of what Step Period you should use for your next part.

GETSP will ask you if it should update the SUPER.PRM Default Paamters File for you. If you answer 'y', then the next part will be made with the calculated step period. You can still override the new step period in the *.L Layer Control File for certain block if you desire.

5.1.2 Editing the Supervisor's Default Parameters File

Here's an example of the new SUPER.PRM file:

```
!
!  Supervisor Parameter File
!  Version 2.21
!
!  last altered: 9/28/87
!        reason: to test new SUPER Default Option line in this file
!

!
!  Default SUPER Options
!
!  place options together inside of quotes;  use "" if no options
!  /NEGZ required for some prototype units
!
"/NEGZ"

!
!  General Parameters
!
800      Elevator board base address
50       Elevator pitch parm, 100 for alphas, 50 for betas
1.0      XY-only scale factor; does not affect Z-axis
0        X-coordinate offset
0        Y-coordinate offset
750      Max number of vectors to send to Stereo

!
!  - Block defaults -
!
!  Each time the Supervisor starts a block, the respective default
!  string below is sent to the Stereo driver.  Z,X... are the block
!  types (Z-Boundary, X-Cross-Hatch, etc.) and they are followed
!  by the parse input file number (e.g. 1 may be for post vectors
!  and 2 for object vectors).  Although only defaults for 2 Slice
!  files are set for here, up to 10 files are supported.
!
#TOP,"BX 8500;BY 34300"          ! laser beam begin (home) position

LB1,"RD 1;RS 300;RC 1;           ! redraw delay, size, count
        SP 25;                   ! step period
        JD 0;                    ! jump delay
        SS 2"                    ! step size
NFDB1,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
LH1,"RC 1;SP 25;JD 0;SS 2;
        VC 5;                    ! rivet count
        VR 99;                   ! rivet reduction amount
        VP 11,12,13,14,15"       ! rivet step periods
NFDH1,"RC 1;SP 25;JD 0; SS 2;VC 5;VR 99;VP 11,12,13,14,15"
FDF1,"RC 1;SP 25;JD 0;SS 16"
```

```
NFDF1,"RC 1;SP 25;JD 0;SS 16"
NFUF1,"RC 1;SP 50;JD 0;SS 16"
FUF1,"RC 1;SP 10;JD 0;SS 16"

LB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
NFDB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
LH2,"RC 1;SP 25;JD 0;SS 2;VC 5;VR 99;VP 11,12,13,14,15"
NFDH2,"RC 1;SP 25;JD 0; SS 2;VC 5;VR 99:VP 11,12,13,14,15"
FDF2,"RC 1;SP 25;JD 0;SS 16"
NFDF2,"RC 1;SP 25;JD 0;SS 16"
NFUF2,"RC 1;SP 50;JD 0;SS 16"
FUF2,"RC 1;SP 10;JD 0;SS 16"

#BTM,"ZW 10;                    ! Z-axis wait time in seconds
      ZD 5;                     ! Z-axis dip depth in mm
      ZV 0.8;                   ! Z-axis velocity parameter
      ZA 0.1"                   ! Z-axis acceleration parameter
```

Blank lines and lines starting with ! are ignored. Text after a ! is also ignored. The first few lines contain fixed Supervisor parameters that can only be affected by changing the numbers here. The rest of the file contains the default settings for the various blocks.

The L1 line contains the default laser control commands sent to STEREO laser driver whenever a L block from (PARSE input) file 1 is encountered.

These default command strings are by no-means fixed. Feel free to change them any way you like, even just to experiment. You can even eliminate all the defaults if they confuse you and start defining them over again.

5.1.3 Editing the Layer Control File

To edit a file, select item 5 from the Main Menu and tell the Editor what file you want to edit. Layer Control Files always have a ".L" extension.

The *.L file (* means any set of file name characters) can contain overrides for any block. Say that we have a FDF1 (Flat Down-Skin Fill from file 1) block. The Supervisor first sends the FDF1 line from its SUPER.PRM file to STEREO:

RC 1;SP 25;JD 0;SS 2

Then, the FDF1 override line from the *.L is sent to STEREO:

SP 10;JD 10

Because the later commands override the earlier ones, the combined effect is:

RC1;SS 2;SP 10;JD 10

```
NFDF1,"RC 1;SP 25;JD 0;SS 16"
NFUF1,"RC 1;SP 50;JD 0;SS 16"
FUF1,"RC 1;SP 10;JD 0;SS 16"

LB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
NFDB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
LH2,"RC 1;SP 25;JD 0;SS 2;VC 5;VR 99;VP 11,12,13,14,15"
NFDH2,"RC 1;SP 25;JD 0; SS 2;VC 5;VR 99;VP 11,12,13,14,15"
FDF2,"RC 1;SP 25;JD 0;SS 16"
NFDF2,"RC 1;SP 25;JD 0;SS 16"
NFUF2,"RC 1;SP 50;JD 0;SS 16"
FUF2,"RC 1;SP 10;JD 0;SS 16"

#BTM,"ZW 10;                    ! Z-axis wait time in seconds
      ZD 5;                     ! Z-axis dip depth in mm
      ZV 0.8;                   ! Z-axis velocity parameter
      ZA 0.1"                   ! Z-axis acceleration parameter
```

Blank lines and lines starting with ! are ignored. Text after a ! is also ignored. The first few lines contain fixed Supervisor parameters that can only be affected by changing the numbers here. The rest of the file contains the default settings for the various blocks.

The L1 line contains the default laser control commands sent to STEREO laser driver whenever a L block from (PARSE input) file 1 is encountered.

These default command strings are by no-means fixed. Feel free to change them any way you like, even just to experiment. You can even eliminate all the defaults if they confuse you and start defining them over again.

5.1.3 Editing the Layer Control File

To edit a file, select item 5 from the Main Menu and tell the Editor what file you want to edit. Layer Control Files always have a ".L" extension.

The *.L file (* means any set of file name characters) can contain overrides for any block. Say that we have a FDF1 (Flat Down-Skin Fill from file 1) block. The Supervisor first sends the FDF1 line from its SUPER.PRM file to STEREO:

```
RC 1;SP 25;JD 0;SS 2
```

Then, the FDF1 override line from the *.L is sent to STEREO:

```
SP 10;JD 10
```

Because the later commands override the earlier ones, the combined effect is:

```
RC1;SS 2;SP 10;JD 10
```

SUPER gives you two levels of control, one in SUPER.PRM, and one in the part's control file. You may want to make special directories for different classes of parts that use different SUPER.PRM files.

NOTE: Future releases of the Supervisor may require alterations to the SUPER.PRM files. Whenever possible, a SUPER.PRM convert file will be supplied.

We recommend modifying only the *.L Layer Control files and leaving the *.P vector files alone. Use the PARSE /Z, /N and /O options to shift layers and exclude certain blocks.

Since both PARSE and SUPER ignore lines that begin with an exclamation point, you can insert comments in the data files:

```
! temporarily reduce step period to test material XQ45
7800,FUF1,"SP5"
```

5.2 Operating the Supervisor

SUPER is the name of the Version 2 Part-Making Supervisor. Version 1's Supervisor was called SUPV and should no longer be used.

When SUPER is brought up, it displays the title:

```
-------------------------------------------
  SUPER, Part Making Supervisor
  3D Systems Laser Stereolithography System
  Version 2.30  by 3D Systems  October 1987
-------------------------------------------

  Part Prefix: _
```

Enter the prefix of the part's file name. SUPER will automatically assume the Layer Control File is the prefix plus ".L" and the vector file is the prefix plus ".P".

SUPER runs much like PARSE in that it display the current Z-layer number and the block name as it runs:

```
  7800: L1 LB1 LH1 NFDB1 NFDH1 FDF1 NFDF1 L2 LB2
  Waiting... Dipping... Relaxing...0
  7810: L1 LB1 LH1 L2 LB2 NFUB2 NFUF2 FUF2
  Waiting... Dipping... Relaxing...19
    :    :    :
```

SUPER tells you when its waiting for the laser controller to idle, when its dipping the elevator, and when its relaxing (delaying). The time to the next layer is displayed after "Relaxing".

Z-stage control

Through the Layer Control File (*.L) and its own SUPER.PRM file, SUPER allows the Stereolithographer to pass along commands to STEREO to control such items as laser beam home position, draw speed, and step size. SUPER has the most flexibility because it doesn't look at the STEREO commands. It just passes the "strings" along to STEREO at the appropriate time.

That control over the XY-laser motion has now been extended to the Z-stage. Four new commands have been added that allow the Stereolithographer to vary various Z-stage parameters:

ZW delay	set the Z-stage wait time (after dipping)
ZD depth	set the Z-stage dip depth (in mm)
ZV velocity	set the Z-stage velocity parameter
ZA accel	set the Z-stage acceleration parameter

These commands can be intermixed with STEREO commands in any way, as in:

```
1800,FUF,"SP 10,10;RC 2;ZW 10;ZD 5;RV 250"
```

This Control Layer File entry sets a new Z-stage wait time and dip depth in the midst of setting new Redraw parameters. The SP and RC commands are passed to STEREO, SUPER analyzes itself the ZW and ZD commands, then the RV command is passed to STEREO. Note that quotes are still required around the whole command string.

5.3 The Supervisor's Options

Like PARSE, SUPER has run-time options that can be appended to the part file name:

/S	-	show stuff
/LOFF	-	laser off
/ZOFF	-	Z-table off

Running SUPER with the /S option will show all the commands SUPER sends to both the Laser Controller (STEREO) and the Z-Table Controller.

/LOFF supresses laser control and /ZOFF supresses Z-table elevator control. Neither /LOFF or /ZOFF affects the output of commands caused by /S. /LOFF and /ZOFF should be used when running the Supervisor without the appropriate control boards installed in the computer.

Negative Z-stage control has also been added to SUPER. "Negative Z-Stage" refers to when the Z-stage works backwards from the default. All the old Beta will require Negative Z-stage control, since positive Z-stage control is defined as proper operation for the new Production Betas. Following "SUPER" or the input file name with "/NEGZ" will cause the Supervisor the use Negative Z-

stage control.

Two more Z-stage related options have been added to SUPER. /START defines at what layer to begin processing, and /STOP defines at what layer to end processing. SUPER will skip past all the beginning layers to the one indicated by /START and end processing just after the layer indicated by /STOP. The layer numbers that follow /START and /STOP are the internal SLICE Z-layer numbers. The following set of options will cause the Supervisor to only process layers 1200 to 1450:

box0035/START 1200/STOP 1450

The following set of options will cause the Supervisor to start processing layers beginning at 2000:

SUPER box0038/START 2000

The spaces after /START and /STOP are not necessary, nor is the case of the letters significant.

5.4 The Supervisor's Keyboard Commands

While the Supervisor is relaxing and counting down the wait period, you can now take one of four actions.

1. You can wait for the wait period to expire, at which point SUPER will begin processing the next layer.

2. You can press P to pause. When the wait period expires, SUPER will wait for a keystroke before continuing. The countdown continues, but the time left is followed by "-P". You can press P again to cancel the pause.

3. You can press C to continue. The rest of the current wait period is skipped, and SUPER begins processing the next layer immediately.

4. You can press S to skip. SUPER aborts the current wait period and asks you what layer you which to skip to. Enter the internal Z layer number and press Enter. SUPER will scan for and resume processing on that layer. If it doesn't find that layer, SUPER will end.

5.5 Using Rivets

The laser controller supports the "rivets", where the middle section of certain vectors are redrawn to help strengthen the plastic part being made. Rivets are a special implementation utilizing the laser controller's redraw mode.

Rivets work as follows:

The redraw pass count is set to 1 using the RC command. The number of redraw passes for the rivets is set with the VC command, and the rivet reduction amount (the amount to chop off of both ends of the rivet vector) is set with the VR command. Then the various step periods for the passes are set using the SP command.

The STEREO Driver processes JX,JY and NX,NY commands as before. (Section 6.3 discusses the STEREO Command Set.) When it encounters a VX,VY sequence, it constructs a rivet on this vector. Internally, this works as follows:

1. STEREO sets the step periods to the VP0 (riVet Period #0) value.
2. STEREO sets up the full vector to be drawn.
3. STEREO sets the redraw pass count to the VC value, the multi-pass step period to VP1, VP2, etc., then sets the reduced vector to be drawn and redrawn.
4. STEREO then restores the original redraw count and the orginal step periods.
5. STEREO continues processing...

The most consistent laser motion occurs with the standard redraw pass count set to 1. The standard redraw mode could be on while rivets are being processed; the standard redraw pass count could be, say, 2, while the rivet redraw pass count could be, say, 6. However, the rivet redraw will truncate the distance of redraw sections just before the rivet, and that may have an effect on rivet performance.

Rivet Implementation

Several STEREO commands support rivets. These commands all begin with 'V' (not 'R', as it is already used for "Redraw"). The code that is responsible for "making rivets" uses a combination of vector moves, jumps, and vector redraws.

VC 1-10	riVet pass Count	(# passes)
VP 5-6553,..	riVet step Periods	(10 us)
VR 0-65535	riVet Reduction amount	(distance)
VX 0-65535	riVet end X position	(position)
VY 0-65535	riVet end Y position	(position)

where:

10 us	is the # of 10s of microseconds resolution = 0.84 us (from 1.19 MHz clock)
position	is the absolute position in the field
distance	is the absolute distance in step size units
# passes	is the number of redraw passes

Like the SP command, the VP command has multiple arguments. Each of up to ten passes of rivet redraw can have its own step period. The VP command can have 1 to 10 arguments, each a step period for a pass:

```
VP vp0,vp1,vp2,vp3...
```

where vp0 is the step period for rivet redraw pass 0 (the full vector draw), vp1 is the step period for redraw pass 1 (the first pass of the rivet), and so on. The VP command is stuffed into the Background Command Queue FOR EACH RIVET, so try to limit the number of rivet redraw passes.

6.0 Other Programs

This chapter discusses some ancillary programs that you may never use. We feel that we should document them anyway just-in-case.

FRONT, CLIFF, and STEREO are documented in this chapter. FRONT is a FRONT end interface to the low-level laser controller, CLIFF is a program that reads STEREO commands from either the keyboard or a file and sends them to STEREO. STEREO is the Stereolithography Device Driver mentioned many times before; its full command set is documented here.

6.1 FRONT, Simple Front End to Laser Controller

FRONT is primarily a debugging tool used in developing the low-level laser controller.

To run FRONT, just type "FRONT" at a DOS prompt. FRONT doesn't care if the laser controller has been loaded or not.

FRONT displays a menu screen that looks like this:

```
          3D Systems Laser Controller Simple Front End
        Version 2.30  by Tarnz Technologies  October, 1987

 A. Laser Installed Check          50. Get Software Interrupt Number
 0. Open Laser Channel             51. Get Background Interval Time
 1. Close Laser Channel            52. Get Laser Control Word
20. Add Vector to Command Queue    61. Set Background Interval Time
21. Add Open Shutter Command       62. Set Laser Control Word
22. Add Close Shutter Command      70. Execute
23. Add Change Time Base Command   71. Execute and Clear
24. Add Pause Command              72. Clear
 3. Reset Command Queue             X. Exit
 4. Get Command Queue Size
```

For the most part, FRONT is self-explanatory. You simple enter the number of the option that interests you at the "FRONT>" prompt and press Enter. FRONT will then ask you for any arguments the option might have. In the case of status options, FRONT will print out the status in a suitable format. If any errors are returned by the low-level laser controller, FRONT will print them out.

6.2 CLIFF, Control Laser Instrument From File

CLIFF stands for "Control Laser Instrument From File" and that is exactly what it does. It reads commands from a text file or optionally from the keyboard and sends them to STEREO Driver.

3D Systems personnel have found it useful to write simple BASIC programs and use CLIFF from the middle of them using the SHELL statement. Their approach to developing laser controlling programs is similar to this procedure:

1. Understand what it is to be accomplished
2. Develop the Basic algorithms and write the code, temporarily displaying intermediate results to the screen for debugging purposes
3. Make a text file filled with STEREO commands
4. Invoke CLIFF by using Basic's SHELL statement
5. See if an error was generated - if the file CLIFF.ERR exists, it contains the error description
6. Dip object being built and process next layer

Somewhere in the BASIC program, the Basic statement

```
SHELL "CLIFF <data file with commands>/T/W"
```

can be found. This is the statement that temporarily suspends execution of the Basic program, executes the string as though it were an MSDOS command, then returns control back to the Basic program at the next statement.

Ordinarily when CLIFF is run, it will display a title, identifying itself and informing you about its version number. The /T option makes CLIFF transparent and forces CLIFF to not output anything to the video screen. This allows the controlling Basic program to assume complete control over the screen.

Ordinarily, CLIFF will end when it translate the last command from the data file. The /W option forces CLIFF, after reading and translating the commands in the text file, to wait until the background task idles before quitting and returning to the BASIC program.

The DOS command line syntax for CLIFF is:

```
CLIFF <datafile>/options
```

where <datafile> is the name of the data file containing the laser control information, and /options are any number of user-selectable options:

```
/S  - show laser control information
/W  - wait for laser idle at end
/T  - transparency mode (no CLIFF text)
```

CLIFF can perform the wait-for-idle function (/W option) without having any data file or commands to execute. CLIFF /W and CLIFF /W/T (transparent) will make CLIFF wait for the backgroupd task to idle.

<datafile> can be replaced by a sequence of direct commands by preceding the commands with a ', ", or @ character. A semicolon can be used to separate commands. As many commands as will fit on one line can be specified. In the following example, the block mode is turned on and the laser beam moved:

```
CLIFF 'MD BL +; JX 1000; JY 2000; NX 4000; NY 5000; EC
```

If CLIFF is called up with no arguments, it will prompt with

```
Enter file name: _
```

at which point the operator should enter the name of the data file containing CLIFF-compatible commands. Direct commands can be entered here if preceded by a special character as described above.

When CLIFF is executed, it erases the CLIFF.ERR file in the currently directory if it exists. Any errors encountered during the processing of the data file will be stored in the CLIFF.ERR file as well as displayed on the screen. This feature allows for the operator to review more that a screenful of error messages, and it allows programs that called CLIFF to see what errors occured.

CLIFF reads a file, line by line, and sends the output to the STEREO device driver. Any errors indicated by STEREO are reported.

6.3 The Stereolithography Device Driver

STEREO is a memory-resident driver specially designed for Sterolithography. Its command set is intended to be somewhat compatible with General Scanning's DG Series Electronic Packages for Laser Control.

STEREO installs itself in the computer's memory when the Process Computer powers up and stays there. Any program can come along and call on STEREO to provide laser controller services.

6.3.1 STEREO Commands

STEREO understands the following commands:

BX 0-65535	Beginning X position	(position)
BY 0-65535	Beginning Y position	(position)
JX 0-65535	Jump to new X position	(position)
JY 0-65535	Jump to new Y position	(position)
NX 0-65535	Next X position	(position)
NY 0-65535	Next Y position	(position)
SS 1-65535	Step Size	(rel incr)
SP 5-6553,..	multi-pass Step Periods	(10 us)
SD 0-65535	Scanning Delay	(step periods)
JS 1-65535	Jump Size	(rel incr)
JD 0-65535	Jump Delay	(step periods)
LO 0-65535	Laser On delay	(10 us)
LF 0-65535	Laser Off delay	(10 us)
NS	eNter Sequence mode	-
AB	Enter Absolute mode	-
EX	Execute	-
EC	Execute and Clear	-
CL	Clear table	-
MD AS -/+	set Auto-Shutter mode	(off/on)
MD BL -/+	set BLock mode	(off/on)
MD PA -/+	set Pos Ack mode	(off/on)
SR	Soft Reset	
HR	Hard Reset	
RC 1-10	Redraw pass Count	(# passes)
RD 0-65535	Redraw Delay	(step periods)
RS 0-65535	Redraw Size	(distance)
VC 1-10	riVet pass Count	(# passes)
VP 5-6553,..	riVet step Periods	(10 us)
VR 0-65535	riVet Reduction amount	(distance)
VX 0-65535	riVet end X position	(position)
VY 0-65535	riVet end Y position	(position)
WI 0-32767	Wait for Idle	(delay)

where:

10 us is the # of 10s of microseconds
resolution = 0.84 us (from 1.19 MHz clock)

position	is the absolute position in the field range in 0 to 65535
distance	is the absolute distance in step size units
rel incr	is the relative # of position increments
# passes	is the number of redraw passes
step periods	is the time to cover that # of steps programmed by SP command (= tick counts)
delay	delay value in milliseconds (may be faster on faster computers)

In the above list, following each two-character command is the range of numbers allowed as arguments to the command. All numbers are in unsigned decimal.

BX and BY set the laser beam's home position. This is the position in the field that the laser beam will be place at when there is no part-making activity.

JX and JY move the laser beam to a new location quickly. No polymerization takes place.

The Jump is conditioned by JS and LD. JS defines the Jump Size, the number of field units to jump at one time. This is usually a large number, like 10000. JD sets the number of step period delays once the jump movement have been completed. This lets the mechanical mirrors settle.

NX and NY move the laser beam to a new location slowly. Polymerization does take place.

The slow laser beam movement is controlled by SS, SP, and SD. SP defines the Step Period, the time for each discrete laser step. SS defines the Step Size, the number of units in the field to move each step period. SD is the delay once the laser beam reaches its destination.

LO and LF set the Laser On and Laser Off delays going into and out of the idle mode. These are delays for the homing and unhomeing of the laser beam.

NS and AB are accepted but not acted upon. They are provided simply for General Scanning compatibility.

On the SLA-1, we usually use MD AS -, MD BL +, and MD PA -. MD BL + turns on the block mode, where laser controller commands are queue up and then let go with the EX or EC command. This technique prevents unnecessary idle times.

EX causes then lase set of commands to be executed. Another EX will cause the same set of commands to be executed again. EC executes, but then clears out the commands. CL clears out the command without executing. With the block mode on, nNo commands will be interpreted unless EX or EC is used.

SR and HR reset the low-level laser controller. HR is the most useful since it stops all current activity and empties out all

the previous commands. HR can be used at the beginning of programs.

RC, RD, and RS control the Redraw parameters. Redraw is where the laser controller makes multiple passes over the same vectors. RC defines the pass count, RD the delay before starting a pass, and RS the size of the redraw. Every RS potion of the vectors will be redrawn if RC is greater than 1.

VC, VP, VR, VX, and VY control the Rivet feature. Discussed in more detail in Section 5.5, rivets are where the interiors of a vectors are redrawn. VC defines the number of redraw passes, VP the multiple step periods, one for each pass, and VR the amount to chop off of both end of the vector. Zero and negative length vectors are not redrawn. To tell STEREO a vector is a rivet, use VX and VY instead of NX and NY.

WI waits for idle. WI's parameter is some delay so that the low-level laser controller will have time to synchronize with STEREO. Typically, WI 1000 will work just fine.

The SP (and VP) command has multiple arguments. Each of up to ten passes of redraw can have its own step period. The SP command can have 1 to 10 arguments, each a step period for a pass:

```
SP sp0,sp1,sp2,sp3...
```

where sp0 is the step period for redraw pass 0 (the original vector move), sp1 is the step period for redraw pass 1, and so on.

The command line with invokes STEREO and causes it to be installed in memory takes the form:

```
STEREO [/options]
```

where /options are any number of user-selectable options:

```
/NV   -  no constant vector velocity
/NG   -  no geometric correction
/NC   -  no version check at install time
```

STEREO can only be installed once after the computer has been powered up. If different options are required, the computer system will have to be completely rebooted.

Geometric correction is automatically invoked. If geometric correction is not desired, the option /NG should be specified in the command line. Geometric correction occurs on the endpoints of any vector STEREO passes to the low-level laser controller.

6.3.2 STEREO Options

LASER and STEREO are installed automatically when the Process Computer starts up. LASER contains the low-level laser controller and has no options. STEREO, on the other hand, has

options. For the most part, these options are of little interest, but are documented for the sake of completeness.

Axes Control

Because we have SLA units with axes going in different dirrections, we installed Axes Control into the STEREO driver. When STEREO is brought up, either by the user typing or from an AUTOEXEC.BAT file, the following options can be use to set the axes:

/X+	X-axis is positive. This is the current default.
/X-	X-axis is negative. Using this flips the direction of the X-axis.
/Y+	Y-axis is positive. This is the current default.
/Y-	Y-axis is negative. Using this flips the direction of the Y-axis.
/SWAPXY	swaps the X and Y coordinates BEFORE any other axis or geometric correction operation. Swapping is not ordinarily done.

Here's a sample DOS command line that installs STEREO with the X-axis direction changed. Note that STEREO must not have already been installed.

STEREO/X-

STEREO reports to you the axes orientation it is going to use when it installs itself. Check this and make sure you got what you asked for.

Geometric Correction

Geometric Correction has been installed in STEREO. Geometric Correction now not only corrects for the pincushion distortion, but also for the imprecisions of each individual SLA unit.

We call the current correction technique Table-Interpolation Geometric Correction because the correction is done after performing look-ups in a special table. This table maps the linear XY coordinate system in a distored one that ends up cancelling the real SLA unit's distortion. The look-up table is stored in a file generated by CALIB, the SLA machine calibration program. STEREO's /NG option will cancel Geometric Correction.

The look-up table has 65 by 65 entries, one entry for each 1024 steps across the X and Y axes plus the end. Remember each axis has a resolution of 65536 steps. The table is stored in the ASCII file STEREO.GEO (see Section 6.7), which is read when STEREO installs itself. If STEREO can't find STEREO.GEO or if it detects some problem with it, it will display an appropriate message.

Each SLA machine will have a different STEREO.GEO based on the results of its calibration. One machine's Geometric Correction will not be the same as that for any other in maintaing the desired mil-accuracy. (The other machine will still run; the parts it makes may seem strange, however.) Think about recalibrating if you move around STEREO.GEO files.

7.0 File Formats

This chapter describes the various Stereolithography file formats. Of these, the ones you will become the most familar with are the Layer Control File (*.L) and the SUPER.PRM Default Parameters File.

7.1 SLICE's Input File (*.STL) Format

A CAD program must be able to produce a file with the specific format described below. Usually this file is quite large is size (several hundred thousand bytes) and is transferred over a high-speed data link such as Ethernet to the 386-based Stereolithography Computer. Transfers of smaller-size files using RS-232 and floppy diskettes are also possible but not recommended.

The SLICE Input Format loosely follows a graphics standard called PHIGS (Programmers Hierarchical Interactive Graphics Standard), and improves on it in several ways. First, all the numeric data can be compressed into binary form, greatly reducing the size of storage files and reducing the data transfer time to the Stereolithography Computer. Second, support of special part-building attributes allow certain features to be "attached" to facets and passed all the way down the pike to the Part-Making Supervisor.

Files being submitted to SLICE should has the extension ".STL", that is, the names of the files should end in ".STL". When SLICE is run and no extension is supplied, SLICE will automatically assumes a ".STL" extension. Older files that has a ".DAT" extension in their file name should either be renamed or have their file names specified completely in SLICE's command line (see Section 2.1).

The files can be stored in either ASCII or Binary. We recommend using the ASCII format while developing STL-file-making software and then switching to the binary format for release. Older versions of SLICE, pre-SLICE17 do not support the binary format.

An example of an ASCII STL file follows. It defines a simple tetrahedron.

```
  solid Solid_Tetra.2
    facet normal -1 0 0
      outer loop
        vertex 0 0 0
        vertex 0 0 1
        vertex 0 1 0
      endloop
    endfacet
    facet normal 0 -1 0
      outer loop
        vertex 0 0 0
        vertex 1 0 0
        vertex 0 1 0
        attribute 16
      endloop
    endfacet
    facet normal 0 0 -1
      outer loop
        vertex 0 0 0
        vertex 0 1 0
        vertex 1 1 0
        attribute 16
        attribute 9
        attribute 22
      endloop
    endfacet
      :       :
  endsolid Solid_Tetra.2
```

The important structure for an ASCII STL file is its use of words to identify the type of numeric data. The facet normal and vertex information can be in floating point - numbers like 43.332382912 and 1.3045E+3 are acceptable.

The vertices should follow the right-hand rule. The normal vector should have a magnitude of 1.

We assume a word is 16 bits and is equal to two bytes. The precise format for a Binary STL file is:

```
(top of file)
80 bytes - general information
           contains part names, comments, etc.

 2 words - number of facet records
           each facet record defines one triangle

First facet record:
 6 words - Normal Vector
            2 words - i coordinate
            2 words - j
            2 words - k
18 words - Triangle Vertices
            2 words - x1 )
            2 words - y1 )-first vertex
            2 words - z1 )
            2 words - x2    )
            2 words - y2    )-second vertex
            2 words - z2    )
            2 words - x3       )
            2 words - y3       )-third vertex
            2 words - z3       )
 1 word  - Number of Attributes
            < special attribute formats >
            This word should be set to zero
    :       :       :
```

The STL Binary Format is similar in structure to the ASCII format. One facet record follows another, and each facet record defines a normal, three triangle vertices, and optionally some attributes. Since we don't as yet support attributes, the attribute count word should be set to zero.

The binary format for the number of facet records and for each number of attributes is just an unsigned integer. The normal and triangle vertices are in four-byte 8087 real format, with a 3 byte mantissa and a 1 byte exponent.

7.2 SLICE's Output File (*.SLI) Format

SLICE produces one output file for each input file. By default, each output file has the extension ".SLI". The current format for the output file is as follows:

```
!
!  optional comments
!
L <layer1>                      L-block (mandatory)
LB                              non-L blocks are only present when
  <X1> <Y1> <X2> <Y2>             they have vectors
  <X1> <Y1> <X2> <Y2>
  <X1> <Y1> <X2> <Y2>             LB   - Layer Border
  <X1> <Y1> <X2> <Y2>
    :    :
LH                                LH   - Layer Hatch
  <X1> <Y1> <X2> <Y2>
    :    :
NFDB                              NFDB - Near-Flat Down-skin Border
  <X1> <Y1> <X2> <Y2>
    :    :
NFDH                              NFDH - Near-Flat Down-skin Hatch
  <X1> <Y1> <X2> <Y2>
    :    :
NFUB                              NFUB - Near-Flat Up-skin Border
  <X1> <Y1> <X2> <Y2>
    :    :
FDF                               FUF  - Flat Down-skin Fill
  <X1> <Y1> <X2> <Y2>
    :    :
NFDF                              NFDF - Near-Flat Down-skin Fill
  <X1> <Y1> <X2> <Y2>
    :    :
NFUF                              NFUF - Near-Flat Up-skin Fill
  <X1> <Y1> <X2> <Y2>
    :    :
FUF                               FUF  - Flat Up-skin Fill
  <X1> <Y1> <X2> <Y2>
    :    :
L <layer2>                      second Z-layer of file
    :    :                        (format repeats)
    :    :
```

where <X1> <Y1> <X2> <Y2> are the two coorinates of a vector in X-Y space, four ASCII numbers
<X1> <Y1> is the start of the vector
<X2> <Y2> is the end of the vector

7.3 The Parser's Output File (*.P and *.L) Format

For a given set of input files, PARSE will produce one *.L file (* represents any set of DOS filename characters) and one *.P file. The *.L file is small and contains layer information, and the *.P file contains the combined vectors.

The *.P file is similar in appearance to the original *.SLI files but differs in three ways.

First, the *.P file is the result of combining all the input files together to form a single vector file for the Supervisor. PARSE had juggled the input files so that all the layers are in ascending order - just the way the Supervisor expects them.

Second, if told to do so, PARSE has selectively included or excluded certain types of blocks, so the vectors for the excluded blocks will not appear in the *.P file.

Third, PARSE has tacked on a file and layer identifier to each block header in the *.P file. Whereas the input line was just:

```
LH
```

PARSE may output to the *.P file:

```
LH2,7800
```

indicating that the block came from the second input file and belongs on layer 7800.

The *.P files can get quite long, so editing them might require a editor that can go beyond 64K of characters. Smaller files of course can be easily edited by virtually any editor.

The *.L files are the most interesting. For each block output to the *.P file, a corresponding single-line entry was made to the *.L file. In addition, a #TOP and a #BTM line-entry wad made for each layer - the #TOP just above and the #BTM just below the layer's other block entries. Here's a sample:

```
7800: L1 LB1 LH1 NFDB1 NFDH1 FDF1 NFDF1 L2 LB2
Waiting... Dipping... Relaxing...0
7810: L1 LB1 LH1 L2 LB2 NFUB2 NFUF2 FUF2

* 1000,50,.508
7800,#TOP
7800,L1
7800,LB1
7800,LH1
7800,NFDB1,"SP 3;RC 1"
7800,NFDH1
7800,FDF1
```

```
  7800,NFDF1
  7800,L2
  7800,LB2
  7800,#BTM
  7810,#TOP
  7810,L1
  7810,LB1
  7810,LH1
  7810,L2
  7810,LB2
  7810,NFUB2
  7810,NFUF2
  7810,FUF2
  7810,#BTM
    :
```

The first line contains the scale, Z-spacing, and layer thickness as told to PARSE. These values are picked up later by the Supervisor.

Note that each line starts with the layer number, followed by the block type. Any line can be followed by:

,"<override commands>"

which contains override commands. The override command is the last STEREO command string sent to the STEREO device driver before drawing the block's vectors, and it can override previously set laser control defaults. Line 7800,NFDB1 above contains such a line. PARSE does not ever make any override commands - only the operator can do that. The override feature is discussed in more detail in Section 5.1.

7.4 Critical Area File (*.BOX) Format

Version 2.04 supports critical areas, where the Supervisor can be instructed to perform or not to perform something within the confines of a box. For a given part with a file name prefix of <part>, PARSE attempts to read the <part>.BOX file. If it doesn't see the file, it simply prints out a warning message and continues. Otherwise, it reads in the file and analyzes it while parsing.

The Format for the *.BOX file is:

```
<type>,<base>,<height>,<x1>,<y1>,<x2>,<y2>,<x3>,<y3>,<x4>,<y4>
   :      :       :      :
```

where:

```
  <type> is 'XV' to rivet cross-hatches in the enclosed area
         or 'XI' to ignore cross-hatches

  <base>    is the base of the box relative to the scale
  <height>  is the height of the box
  <x1>,<y1> is the first coordinate of the box
  <x2>,<y2> is the second coordinate
  <x3>,<y3> is the third coordinate
  <x4>,<y4> is the fourth coordinate
```

If the CAD image was drawn in mils (1000ths of inches) and the scale factor is 1000, then:

```
  XV,8.0,0.3,1,1,1,1,-1,-1,-1,-1,1
  XI,7.90,0.2,2,2,2,2,-0.5,-3,-2,-1,3.1
```

defines two boxes, one 3/10 and one 2/10 inches high. The first box tells where cross-hatching with rivets will occur, and the second box tells where no cross-hatching will occur. All the coordinates are expressed in inches.

The current box algorithm supports only lines on the X and Y-axes. Only rectangles can be specified. Don't specify diamonds or near-triangles. If you need to isolate a precise area, use a set of rectangles that nearly approximate it.

7.5 Supervisor Default Parameters File (SUPER.PRM) Format

Blank lines and lines starting with ! are ignored. The first few lines contain fixed Supervisor parameters that can only be affected by changing the numbers there. The rest of the file contains the default settings for the various blocks.

```
!
!  Supervisor Parameter File
!  Version 2.30
!
!  last altered: 9/28/87
!        reason: to test new SUPER Default Option line in this file
!

!
!  Default SUPER Options
!
!  place options together inside of quotes;  use "" if no options
!  /NEGZ required for some prototype units
!
""

!
!  General Parameters
!
800      Elevator board base address
50       Elevator pitch parm, 100 for alphas, 50 for betas
1.0      XY-only scale factor; does not affect Z-axis
0        X-coordinate offset
0        Y-coordinate offset
750      Max number of vectors to send to Stereo

!
!  - Block defaults -
!
!  Each time the Supervisor starts a block, the respective default
!  string below is sent to the Stereo driver.  Z,X... are the block
!  types (Z-Boundary, X-Cross-Hatch, etc.) and they are followed
!  by the parse input file number (e.g. 1 may be for post vectors
!  and 2 for object vectors).  Although only defaults for 2 Slice
!  files are set for here, up to 10 files are supported.
!
#TOP,"BX 8500;BY 34300"          ! laser beam begin (home) position

LB1,"RD 1;RS 300;RC 1;           ! redraw delay, size, count
        SP 25;                   ! step period
        JD 0;                    ! jump delay
        SS 2"                    ! step size
NFDB1,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
LH1,"RC 1;SP 25;JD 0;SS 2;
        VC 5;                    ! rivet count
        VR 99;                   ! rivet reduction amount
```

```
        VP 11,12,13,14,15"      ! rivet step periods
NFDH1,"RC 1;SP 25;JD 0; SS 2;VC 5;VR 99;VP 11,12,13,14,15"
FDF1,"RC 1;SP 25;JD 0;SS 16"
NFDF1,"RC 1;SP 25;JD 0;SS 16"
NFUF1,"RC 1;SP 50;JD 0;SS 16"
FUF1,"RC 1;SP 10;JD 0;SS 16"

LB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
NFDB2,"RD 1;RS 300;RC 1;SP 25;JD 0;SS 2"
LH2,"RC 1;SP 25;JD 0;SS 2;VC 5;VR 99;VP 11,12,13,14,15"
NFDH2,"RC 1;SP 25;JD 0; SS 2;VC 5;VR 99;VP 11,12,13,14,15"
FDF2,"RC 1;SP 25;JD 0;SS 16"
NFDF2,"RC 1;SP 25;JD 0;SS 16"
NFUF2,"RC 1;SP 50;JD 0;SS 16"
FUF2,"RC 1;SP 10;JD 0;SS 16"

#BTM,"ZW 10;                    ! Z-axis wait time in seconds
      ZD 5;                     ! Z-axis dip depth in mm
      ZV 0.8;                   ! Z-axis velocity parameter
      ZA 0.1"                   ! Z-axis acceleration parameter
```

The file lets the STEREO commands be intermixed with the special Supervisor Z-stage commands. Command strings can be broken up into several lines, and comments (beginning with an !) can be at the end of any line.

7.6 The STEREO.DEF Driver Default File Format

Below is a copy of the STEREO.DEF file supplied with the system. These are the default values loaded by STEREO when it is installed. You may never need to change any of these values in this file. In fact, the Supervisor always resets these parameters using the appropriate lines from the SUPER.PRM and *.L files.

```
!
!  STEREO Device Driver Initial Set-up
!
2       [SS] Step Size
1190    [SP] Step Periods
64      [SD] Scan Delay
65535   [JS] Jump Size
100     [JD] Jump Delay
0       [LO] Laser On Delay
0       [LF] Laser Off Delay
1       [RC] Redraw Pass Count
0       [RD] Redraw Delay
200     [RS] Redraw Size
2       [VC] Rivet Pass Count
1190    [VP] Rivet Step Periods
100     [VR] Rivet Reduction Amount
```

7.7 The STEREO.GEO Geometric Correction File Format

The geometric correction look-up table has 65 by 65 entries, one entry for each 1024 steps across the X and Y axes plus the end. Remember each axis has a resolution of 65536 steps. The table is stored in the ASCII file STEREO.GEO in the \3DSYS directory, which is read when STEREO installs itself. If STEREO can't find STEREO.GEO or if it detects some problem with it, it will display an appropriate message.

So, the format for the look-up table is as follows:

```
!
!  Comment Lines (optional)
!

65,65,2           ! 65 by 65 by 2 coordinates
102.34,134.56     ! correction for (0,0)
101.23,123.45     ! correction for (0,1024)
   :       :       :
102.41,144.33     ! correction for (0,65536)
98.767,102.22     ! correction for (1024,0)
96.352,100.034    ! correction for (1024,1024)
   :       :       :
```

Note that even though we are mapping onto an integer-system, the correction table can have floating-point numbers. The first non-coment line must contain "65,65,2" as that is the only geometric correction format currently supported by STEREO.

Each SLA machine will have a different STEREO.GEO based on the results of its calibration. One machine's Geometric Correction will not be the same as that for any other in maintaing the desired mil-accuracy. DO NOT COPY STEREO.GEO FILES FROM ONE SLA MACHINE TO ANOTHER.

Appendix A - SLA-1 Software System Chart

- Process Computer -	- Slice Computer -
Start-Up Installation	
MSDOS	XENIX
TCP	TCP
LASER	
STEREO	
Menu Programs	
MENU	
System Functions	
CHKDSK	SYSADM
DIR	
DOS	
QD2	
Slicing Programs	
	UI
	SLICE
Part-Making Programs	
PARSE	
SUPER	
GRAPH	
Utility Programs	
POWER	
GETSP	
ZSTAGE	
CALIB	
PROFILE	
Communication Programs	
TELNET	TELNET
FTP	FTP

Appendix B - The STEREO Command Set

STEREO Command Set is intended to be somewhat compatible with General Scanning's DG Series Electronic Packages for Laser Control.

STEREO understands the following commands:

BX 0-65535	Beginning X position	(position)
BY 0-65535	Beginning Y position	(position)
JX 0-65535	Jump to new X position	(position)
JY 0-65535	Jump to new Y position	(position)
NX 0-65535	Next X position	(position)
NY 0-65535	Next Y position	(position)
SS 1-65535	Step Size	(rel incr)
SP 5-6553,..	multi-pass Step Periods	(10 us)
SD 0-65535	Scanning Delay	(step periods)
JS 1-65535	Jump Size	(rel incr)
JD 0-65535	Jump Delay	(step periods)
LO 0-65535	Laser On delay	(10 us)
LF 0-65535	Laser Off delay	(10 us)
NS	eNter Sequence mode	-
AB	Enter Absolute mode	-
EX	Execute	-
EC	Execute and Clear	-
CL	Clear table	-
MD AS -/+	set Auto-Shutter mode	(off/on)
MD BL -/+	set BLock mode	(off/on)
MD PA -/+	set Pos Ack mode	(off/on)
SR	Soft Reset	
HR	Hard Reset	
RC 1-10	Redraw pass Count	(# passes)
RD 0-65535	Redraw Delay	(step periods)
RS 0-65535	Redraw Size	(distance)
VC 1-10	riVet pass Count	(# passes)
VP 5-6553,..	riVet step Periods	(10 us)
VR 0-65535	riVet Reduction amount	(distance)
VX 0-65535	riVet end X position	(position)
VY 0-65535	riVet end Y position	(position)
WI 0-32767	Wait for Idle	(delay)

where:

10 us	is the # of 10s of microseconds resolution = 0.84 us (from 1.19 MHz clock)
position	is the absolute position in the field range in 0 to 65535
distance	is the absolute distance in step size units
rel incr	is the relative # of position increments
# passes	is the number of redraw passes
step periods	is the time to cover that # of steps programmed by SP command (= tick counts)

delay	delay value in milliseconds (may be faster on faster computers)

Section 6.3.1 discusses these commands in more detail.

Appendix C - Stereolithography Software Glossary

Below, we present a sofware glossary of terms you may have encountered in this documentation or may hear in the future. Please take a few minutes to review these terms.

80286 - the 16-bit microprocessor in the Process Computer.

80386 - the 32-bit microprocessor in the Slice Computer.

block - a set of vectors in a file

critical area - an area where vectors will be treated special

cross-hatching - supporting cross-vectors between the inside and outside walls of a part

Ethernet - a high-speed intra-computer communications system jointly developed by Digital, Intel and Xerox

field - the surface of the vat that the laser beam can reach

filling - the process by which part surfaces are made

geometric correction - adjusting the laser beam for the pincushion effect

home position - the field location that the beam jumps to when there is no part-making activity

laser controller - the software "underneath" the Supervisor that controls the laser's motion

jump - the quick, non-polymerization moving of the laser beam

jump delay - the delay after a jump allowing the mirrors to settle

mirrors - those which rotate to position the laser beam anywhere in the field

MSDOS - the standard Disk Operating System for the Process Computer

multi-tasking - where many people can use the same computer simultaneously

pincushion effect - the inherent distortion of a pin-point source being directed onto a flat surface

process - the act of making a part

Process Computer - the computer in the SLA machine that is used to make parts; MSDOS-based

redrawing - making multiple passes of the laser beam over the same vector to induce strength

rivets - an intelligent method of redrawing where the extra passes are made over a smaller area

segment - two-coordinate, four-point describing a line

shutters - barriers of the laser beam

slicing - the act of preparing a 3D part for Stereolithography by transforming it into 2D layers

Slice Computer - the computer away from the SLA machine that is use to slice parts; XENIX-based

step - a discrete movement of the laser beam

step period - the period during which one step is made

step size - the number of field units for a step

unix - an advanced multi-tasking operating system developed at Bell Labs in the 70's

XENIX - a unix-like Operating System provided on the Slice Computer

Z-stage - the vectical elevator parts rest on

Appendix D - Editor Commands

The Editor Commands are similar to those of Word-Star, a popular Word Processing System, and to those found in the Borland International family of products.

Basic Cursor Movement Commands

character left	Ctrl-S or Left
character right	Ctrl-D or Right
word left	Ctrl-A
word right	Ctrl-F
line up	Ctrl-E or Up
line down	Ctrl-X or Down
scroll up	Ctrl-W
scroll down	Ctrl-Z
page up	Ctrl-R or PgUp
page down	Ctrl-C or PgDn

Quick Cursor Movement Commands

beginnning of line	Ctrl-Q S or Home
end of line	Ctrl-Q D or End
top of window	Ctrl-Q E
bottom of window	Ctrl-Q X
top of file	Ctrl-Q R
bottom of file	Ctrl-Q C
beginning of block	Ctrl-Q B
end of block	Ctrl-Q K
last cursor position	Ctrl-Q P

Insert and Delete Commands

insert mode on/off	Ctrl-V or Ins
insert line	Ctrl-N
delete line	Ctrl-Y
delete to end of line	Ctrl-Q Y
delete character to left of cursor	Ctrl-H or Backspace
delete character under cursor	Ctrl-G or Del
delete word right of cursor	Ctrl-T

Block Commands

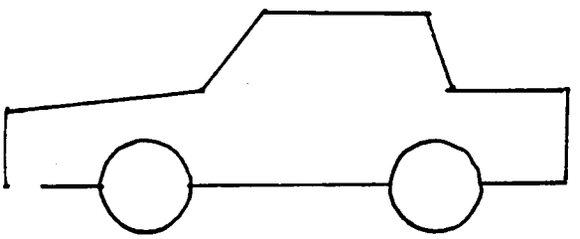
mark block-begin	Ctrl-K B
mark block-end	Ctrl-K K
mark single word	Ctrl-K T
copy block	Ctrl-K C
delete block	Ctrl-K Y
hide/display block	Ctrl-K H
move block	Ctrl-K V
read block from disk	Ctrl-K R
write block to disk	Ctrl-K W

Miscellaneous Commands

find	Ctrl-Q F
find and replace	Ctrl-Q A
repeat last find	Ctrl-L
abort operation	Ctrl-U
restore line	Ctrl-Q L
quit edit	Ctrl-K D

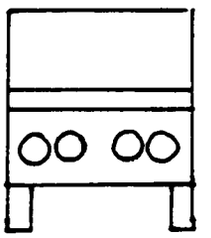
3D Systems Software Manual Addendum #1

SLICE REVEALED TO EVERYBODY



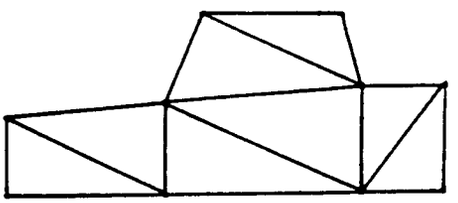
GENERIC AMERICAN AUTOMOBILE, CIRCA 1973

SIDE VIEW (NO DETAIL)

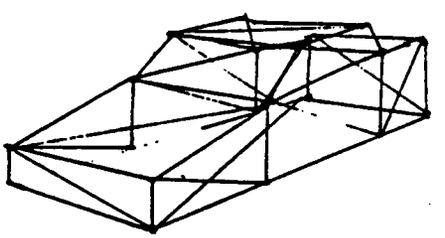


FRONT VIEW (NO DETAIL)

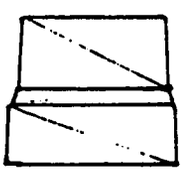
THE GEOMETRIC MODEL SLICE WOULD GET AS INPUT: (NO TIRES)



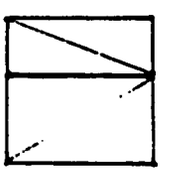
SIDE



(IT'S ALL TRIANGES)

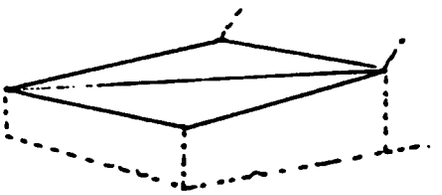


FRONT (NO TIRES)

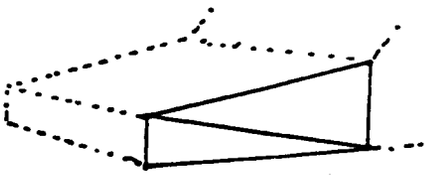


BACK

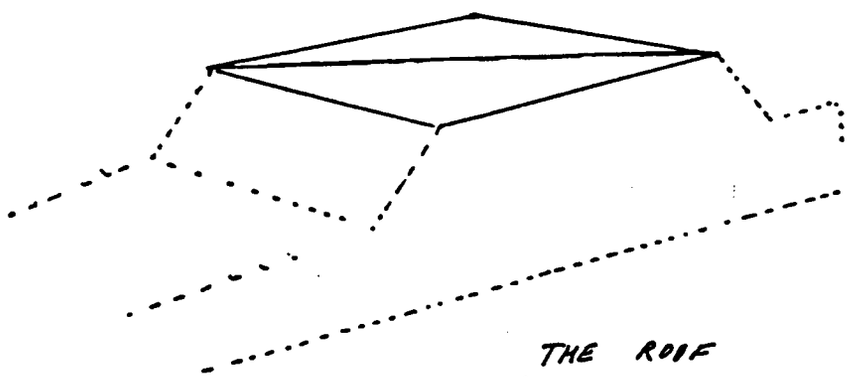
EXPLODED VIEW



THE HOOD
(2 TRIANGES —
THEY ARE "NEAR FLAT")



THE FENDER
(2 TRIANGES —
THEY ARE "SCAN"
TRIANGES)

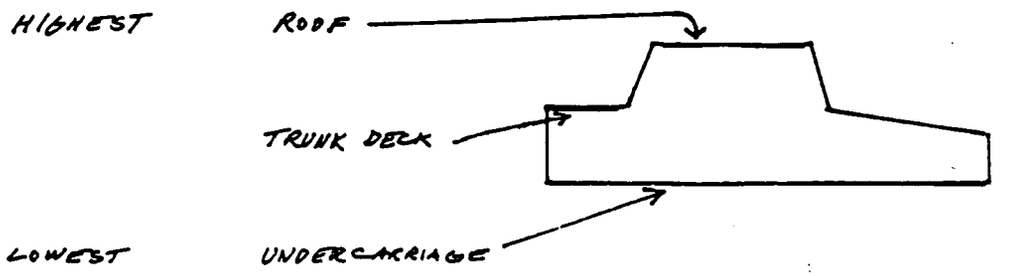


THE ROOF
(2 TRIANGLES —
THEY ARE "FLAT")

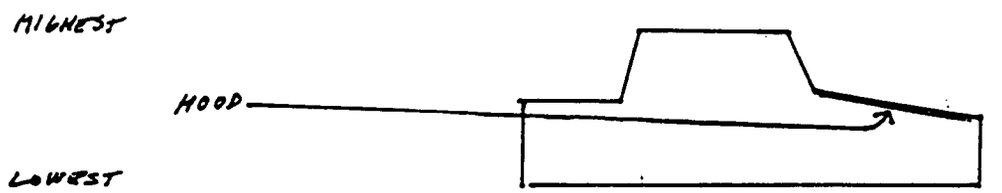
ALL THE OTHER TRIANGLES THAT COMPRISE THE CAR MODEL MUST BELONG TO ONE OF THESE TYPES, FLAT, NEAR-FLAT, OR SCAN.

SLICE ORDERS THE TRIANGLES BY TYPE AND HEIGHT

THE FLAT TRIANGLES

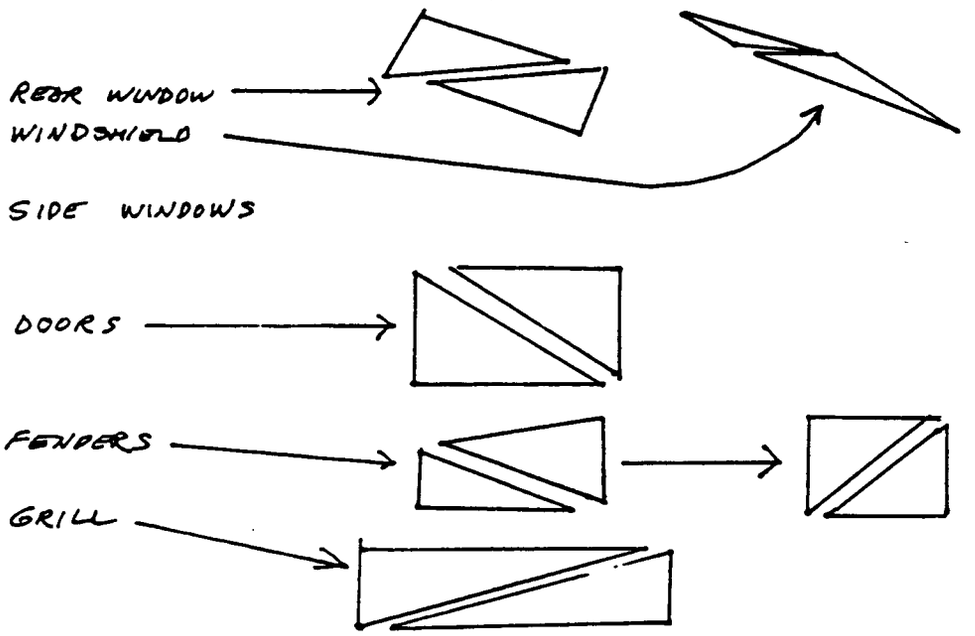


THE NEAR-FLAT TRIANGLES



. . . AND THE SCAN TRIANGLES ARE ORDERED BY THEIR LOWEST CORNER

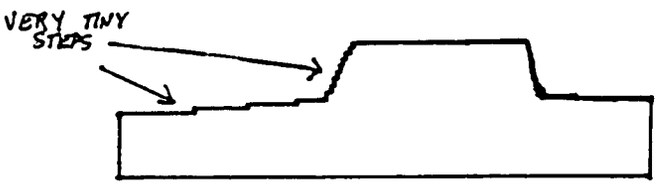
HIGHEST



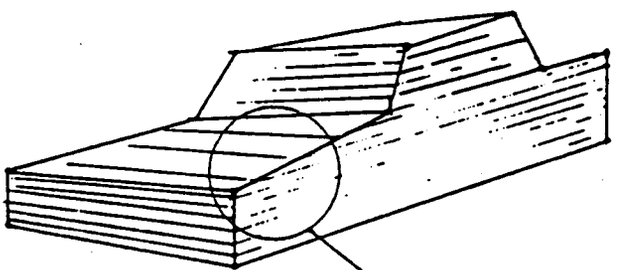
(ACTUALLY, DOORS, FENDERS, AND GRILL TRIANGLES COULD HAVE THE SAME LOWEST CORNER, BUT WE SHOW THEM HERE FOR CLARITY).

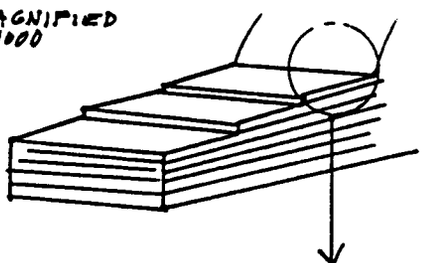
LOWEST

NOW, WE WANT TO PROVIDE THE LASER CONTROLLER SO IT CAN MAKE AN OBJECT LIKE THIS . . .



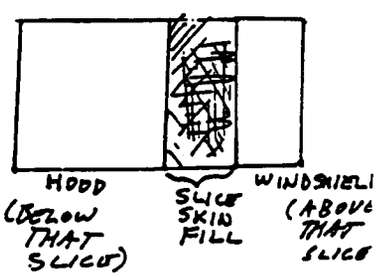
AS IF WE HAD TO SLICE THE CAR AND STACK THE SLICES.



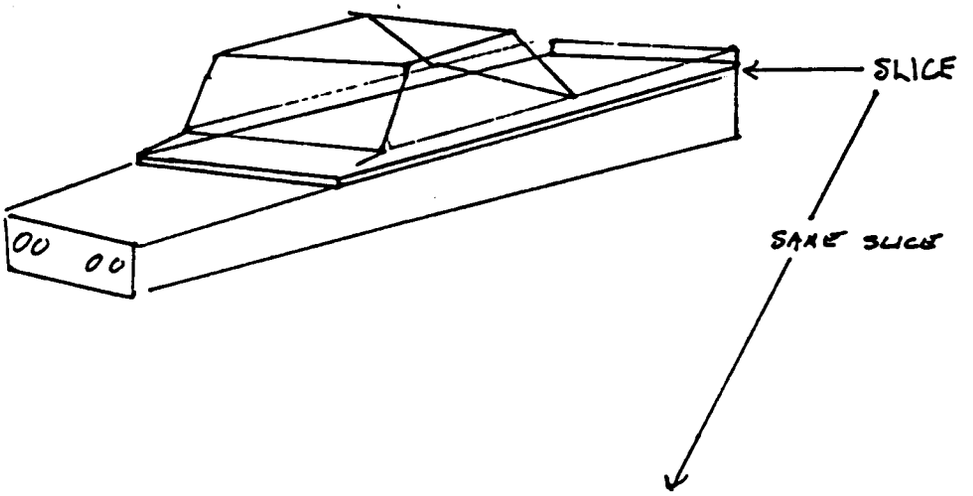


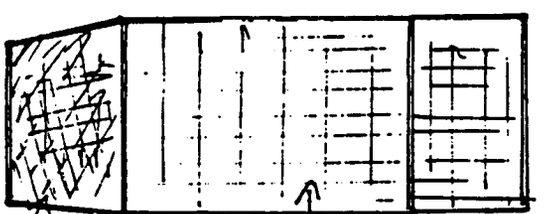
THE NEAR-FLAT TRIANGLES AND FLAT TRIANGLE CAN BE "SKIN-FILLED"

FILLING IS A VERY FINE CROSS HATCHING BY THE LASER TO MAKE THE SURFACE SMOOTH



HOW IS A SLICE FORMED ?





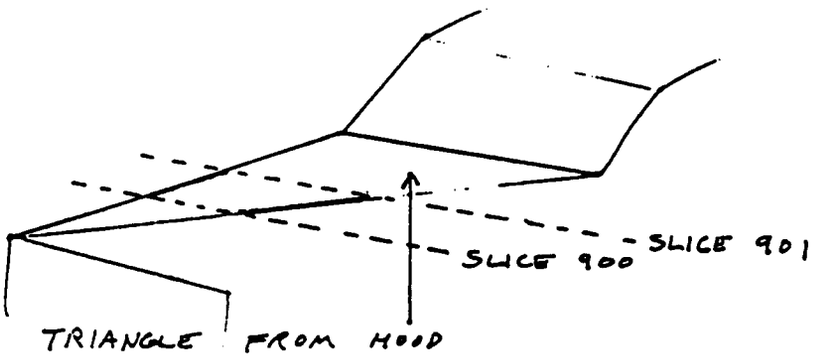
"BORDER" SEGMENTS ENCLOSE THE PERIMETER OF THE SLICE

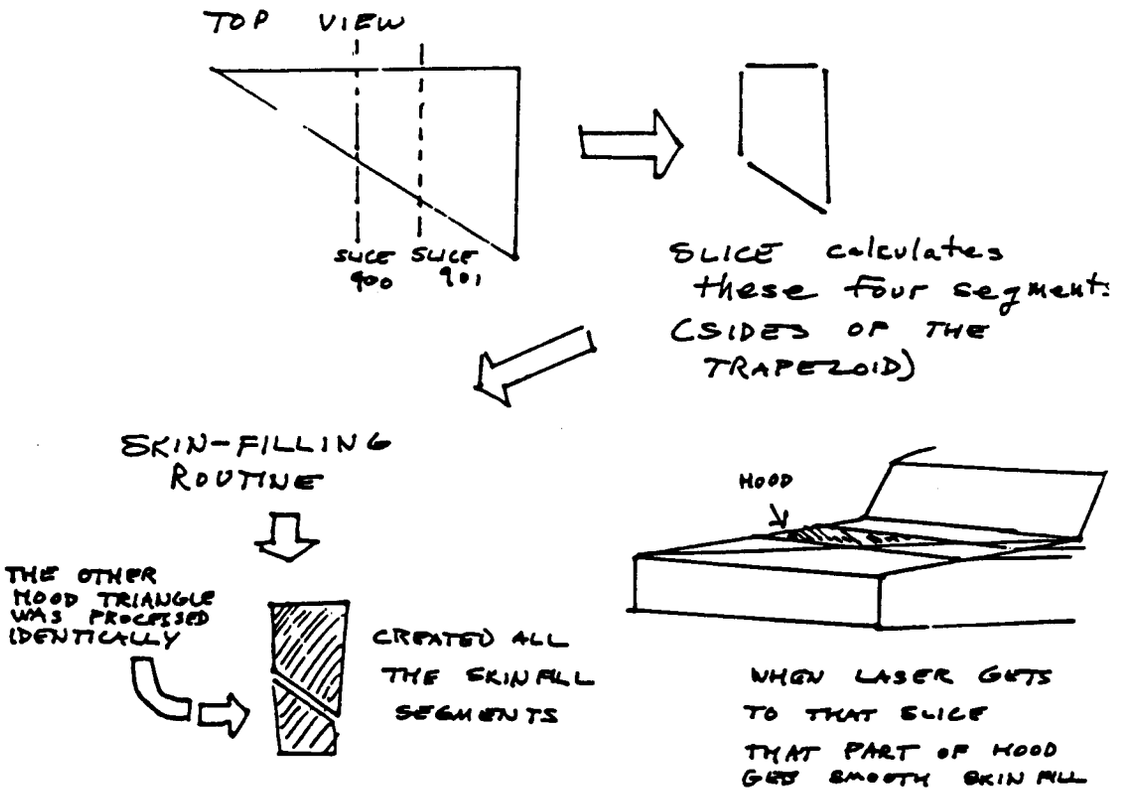
THIS PART GETS SKIN FILLED BECAUSE IT IS "NEAR-FLAT"

INTERIOR BETWEEN BORDERS GETS CROSS MATCHED SO THE SLICE HAS SOME STRENGTH

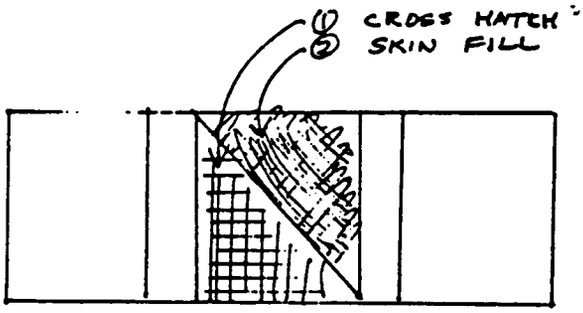
THE SLICE CROSS SECTION IS RECTANGULAR ONLY BECAUSE OUR CAR WAS A BOXY DESIGN.

FOR THE NEAR-FLAT TRIANGLES SLICE MAKES TRAPEZOIDS:





FLAT TRIANGLES ARE CROSS HATCHED AND SKIN FILLED TOO — BUT THERE IS NO NEED TO MAKE TRAPEZOIDS BETWEEN SLICES:

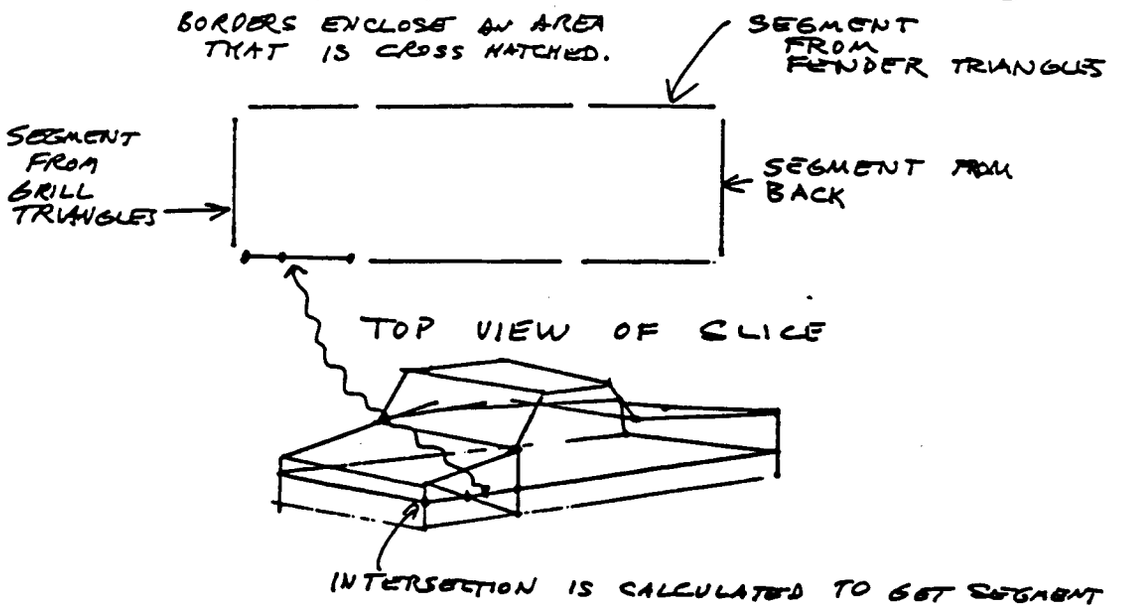


ROOF TRIANGLES (FLAT)

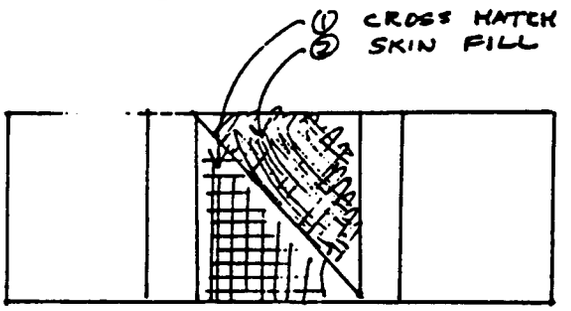
TOP VIEW OF CAR

SCAN TRIANGLES ONLY GENERATE BORDERS: (WHICH ARE THEN CROSS HATCHED)

BORDERS ENCLOSE AN AREA THAT IS CROSS HATCHED.



FLAT TRIANGLES ARE CROSS MATCHED AND SKIN FILLED TOO — BUT THERE IS NO NEED TO MAKE TRAPEZOIDS BETWEEN SLICES:

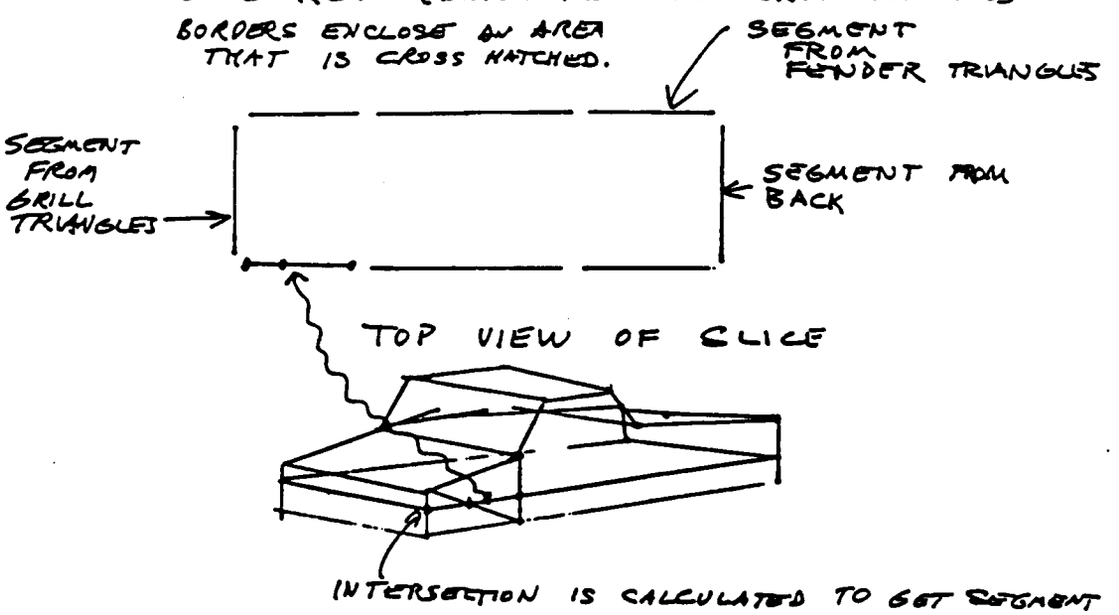


ROOF TRIANGLES (FLAT)

TOP VIEW OF CAR

SCAN TRIANGLES ONLY GENERATE BORDERS: (WHICH ARE THEN CROSS MATCHED)

BORDERS ENCLOSE AN AREA THAT IS CROSS MATCHED.



APPENDIX E

©1989 3D SYSTEMS, INC.
ALL RIGHTS RESERVED

Non-3D Systems Software Status as of 4/13/88

Slice Computer (NEC386)

TM

386/ix Operation System

UNIX; System V Release 1.0.4 80386

TCP Ethernet Support, MICOM Version

INTERACTIVE Systems Corporation

2401 Colorado Avenue, 3rd Floor

Santa Monica, California 90404

Process Computer (WYSE 286)

MS-DOS 3.21

Wyse Technology

3571 N. First Street

San Jose, CA 95134

Q-DOS II Version 2.00

Gazelle Systems

42 North University Avenue, Suite 10

Provo, Utah 84601

FTP Software PC/TCP File Transfer Program Version 1.16

FTP Software PC/TCP Telnet Version 1.16

MICOM-Interlan, Inc.

155 Swanson Rd.,

Boxborough, MA 01719

April 13, 1988

Major parts list for SLA1

Laser:

1) Liconix

1390 Borregas Avenue

Sunnyvale, CA 94089

A) Model 4240H HeCd Multimode Laser

B) Model 4240PS Power Supply

2) Omnichrome

13620 Fifth Street

Chino, CA 91710

A) Model 356XM HeCd Laser

B) Model 100 Power Supply

Scanning mirrors:

1) General Scanning, Inc.
500 Arsonal Street
Watertown, MA 02172
A) P/N E00-Z2173 XY0507 Scanning Mirrors
B) P/N E00-DX2005 Controller for XY0507 Scanning Mirrors

Z-axis (Vertical) Elevator:

1) Daedal
P.O. Box G
Harrison City, PA 15838
(Purchased Via Rep.)
Pacific Technical Products
15901 Foothill Blvd.
Sylmar, CA 91342
A) P/N 008-0324-3 14" Linear Table with 5 Pitch
B) P/N MC5000-20 Motor Control with Motor Drive

SLA1 Process Computer:

1) Wyse Technology
3571 N. First Street
San Jose, CA 95134
(Purchased Via Rep.)
Peripheral Systems, Inc.
8107 Orion Avenue
Van Nuys, CA 91406
A) Wyse 286, Model 2200, includes
1. 40 MByte Hard Disc
2. "AT" Keyboard
3. Monitor
4. Graphic Card
5. Math Coprocessor

2) Tarnz Technologies
8025 Sepulveda Blvd
Van Nuys, CA 91411
A) I/O board

SLA1 Slice Computer:

1) NEC Information Systems, Inc.
1414 Massachusetts Avenue
Boxborough, MA 01719
(purchased Via Rep.)
Peripheral Systems, Inc.
8107 Orion Avenue
Van Nuys, CA 81406
A) NEC Powermate 386 Computer

Beam Expander:

1) Optomeo Design Co.
901 18th St., Suite 203
Los Alamos, N.M. 67544
A) Model # D10493 (4X) Bean Expander

APPENDIX F

©1989 3D SYSTEMS, INC.
ALL RIGHTS RESERVED

VERSION 3.03

==========

3D Systems Stereolithography System Software

==========

BEAM.PAS

SLA-1 Beam Profile and Calibration Software

©1988 3D SYSTEMS, INC.
ALL RIGHTS RESERVED

SLA Software Release 3.0 4/15/88

Program	File Name(s)
BEAM	BEAM.PAS
	BEAMGR.INC
	BEAMIO.INC
	BEAMSCAN.INC
	BEAMUTIL.INC
	BLDMON.INC
BUILD	BUILD.PAS
	LASER.INC
	LCTRL.INC
	GRAPH.INC
	KEYBRD.INC
	TIME.INC
	UTILITY.INC
	ZSTAGE.IND
CALOUT	CALOUT.BAS
CLIFF	CLIFF.PAS
LASER	LASER.ASM
MATERIAL	MATERIAL.BAS
MENU	MENU.PAS
MERGE	MERGE.PAS
MONITOR	MONITOR.PAS
SLICE300	PFP.H
	PFPCHL.H
	PFPCURE.H
	PFPHI.H
	PFPIDB.C

	PFPRIE.C
	PFPRIF.C
	PFPSEG.C
	PFPSEG.H
	PEPTRI.C
	PEPTRI.H
	PFPTUTL.C
	PFPUTL.C
	PFPUTL.H
	PFPZACT.C
	PRTUTL.C
POWER	POWER.PAS
PREPARE	PREPARE.BAS
SERVICE	SERVICE.PAS
SLICE	SLICE.C
	SLICE.H
STEREO	STEREO.PAS
	STERCMDS.INC.
TESTPART	TESTPART.BAS
XFER	XFER.PAS
ZSTAGE	ZSTAGE.PAS
	COMBINE.INC.

```
program Beam;

uses Crt,
     Dos,
     Graph,
     Drivers

type
  str6  = string[6];
  str10 = string[10];
  str16 = string [16];
  OpModeType = (TextOp,GraphicsOp)
  BeamProfileVarSensorType = record
    Xpos             : LongInt;  { inputs }
    Ypos             : LongInt;
    ScanStep         : Integer;
    BkgndThreshold   : Integer;
    InitialDelay     : Integer;
    RowToRowDelay    : Integer;
    ColToColDelay    : Integer;
    XYweight         : LongInt;  { outputs }
    Xweight          : LongInt;
    Yweight          : LongInt;
    TotalWeight      : LongInt;
    HiPoints         : Word;
    SaturationCount  : Integer;
    MaxSensorReading: Interger;
    BeamMap          : array[0..30.0..30] of Integer;
    RowWeight        : array[0..30] of Word;
    ColWeight        : array[0..30] of Word;
  end;
  BeamProfileVarType = array[1..3] of BeamProfileVar
    SensorType;
  FindBeamVarSensorType = record
    Xval             : LongInt;  { inputs & updated }
    Yval             : LongInt;
    SpiralDelay      : Word; { inputs }
```

```
    SpiralSideMax    : Integer;
    SpiralTries      : Integer;
    ScanStep         : Integer;
    BkgndThreshold   : Integer;
    NumCenterPasses  : Integer;
    Xavg             : Real; { outputs }
    Yavg             : Real;
  end;
  FindBeamVarType = array[1..3] of
FindBeamVarSensorType;
  FindMethodType = (none,NormalFind,Verify2MilHole);
  PCM_Mode = (no_PCM,GainDrift,StraightXY);
  PMM_Mode = (no_PMM,Fast,Slow);

const
  VersionId = '3.03';
  SoundOn = false;
  ParamFileName = 'BEAM.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  HomeDirectory = 'C:\3DSYS\';
  CalibFileName = CALIB.DAT';
  Blank = '
  ColMax = 22;
  RowMax = 22;
  PowerMvAvg = 10;

  CtrlC = ^C;
  Bs = ^H;
  Cr= ^M;

  CalibSelect:  array[1..3] of str6 = (' 1 ', '1 or 2',
'1 - 3 ');
  LaserAddr = $300;
  NumSensors = 3;
  DiscreteBeam = 5;
  Timeout = 60000;
  lambda = 5.9;
```

```
var
  FirstVar:  Byte;  { must be first variable defined }
  { beam arrays }
  Resistance,Temperature: array{0..100] of Real;
  { sensor arrays }
  SensorChannel,SensorBkgnd: array [1..3] of Word;
  XSensorPos,YSensorPos  :  array [1..3] of Real;
  Power,Bk: array[1..3] of Real;
  Xsp,Ysp: araray[1..3] of Real;
  SensorCalib: array[1..3,1..4] of Real;
  LastField: array[1..4] of Integer;
  { miscellaneous }
  key: Char;
  PrintDev,CalOutFile,ErrorFile,ParamFile: Text;
  ZTableOff,LaserOff,ShowStuff,DemoMode: Boolean;

GrInstalled,Debug,AbortFlag,PrintOut,Display,Calibrate:
    Boolean;
  BeamFound: Boolean;
  OutFileName,SingleBar,CmdLine: String;
  XRest,YRest: Word;
  TempChannel,TempCalPts: Word;
  PeakReading,Sensor,Select,SensorNum,PosChkSpeed:
Integer;
  Xpos,Ypos: LongInt;
  Xaverage,Yaverage,TotWtAvg: Real;
  GraphDriver,GraphMode,Error: Integer;
  ViewPort: ViewPortType;
  OpMode: OpModelType;
  OldExitProc: Pointer;
  BeamProfileVariables: BeamProfileVarType;
  FindBeamVariables: FindBeamVarType;
  Display1Sensor,Focus5Sensor,GoodCount: Integer;
  CurrentDate,CurrentTime: String;
  Sec,Minu,Hour,Day,WeekDay,Mon,Year: Integer;
  CalibDelays: array[1..3,1..3] of LongInt;
  PositionCheckMode: PCM_Mode;
```

```
  PositionMethodMode: PMM_Mode;
  LastVar: Byte; { should be last variable defined }

procedure Abort; forward;

{-------------------------------------------------------}
{$R-,S-,I+=,F-,V-, B-}
{$M 16384,0,655360 }
{$i beamutil.inc}
{$  beamgr.inc.}
{$i beamio.inc.}
{$i beamscan.inc}

{-------------------------------------------------------
}

{
  check for BEAM Program Options

}
procedure CheckForOptions(var CmdLine: String);
var
  i: Integer
begin
  CmdLine := upper (CmdLine);
  i := pos ('/DEBUG',CmdLine);
  if i>0 then begin
    Debug:= true;
    delete{CmdLine,i,6);
  end;
  i := pos ('/TEXT',CmdLine);
  if i>0 then begin
    OpMode := TextOp;
    delete(CmdLine,i,5);
  end;
  i := pos ('/DEMO',CmdLine);
  if i>0 then begin
```

```
    DemoMode := true;
    delete(CmdLine,i,5);
  end;
  i := pos('/S',CmdLine);
  if i>0 then begin  { check for /S Show Stuff flag }
    ShowStuff := true;
    delete(CmdLine,i,2);
  end;
  i := pos('/LOFF',cMDlINE); {check for/LOFF No Laser
flag}
  if i>0 then begin
    LaserOff := true;
    delete(CmdLine,i,5);
  end;
end;

{
  Read BEAM.PRM parameters file
}
procedure ReadBeamParamsFile;
var
  ParamFile: text;
  i,count,code,code1,code2,Sensor,Pinhole: Integer;
  cmd,ParamLine: String;

  procedure GetParamFileLine;
  begin
    if eof(ParamFile) then ParamLine := ''
    else repeat
      readln(ParamFile,ParamLine);
    until eof(ParamFile) or ((ParamLine <> '') and
      (ParamLine[1] <> '!'));
  end;

begin
  Center('--- Reading Parameters File ---',23);
  {$I-}
```

```
assign(ParamFile,ParamFileName);
reset(ParamFile);
if IOresult<>0 then begin
  assign(ParamFile,HomeDirectory+ParamFileName);
  reset(ParamFile);
  if IOresult<>0 then begin
    Center ('Can''t find Beam Parameter file.',24);
    WaitKey;
    halt(3)
  end;
end;
{$I+}
GetParamFileLine;
if copy(ParamLine,1,1) = chr(34)) then begin { handle
  quoted options line }
  i := pos(chr(34),copy(ParamLine,2,255));
  if i>0 then ParamLine[0] :=chr(i);
  ParamLine := NoQuotes(ParamLine);
  CheckForOptions(ParamLine);
  GetParamFileLine;
end;
XRest := WordVal(ParamLine,code);
YRest := WordVal(copy(ParamLine,code+1,255),code);
GetParamFileLine;
SensorNum := WordVal(ParamLine,code);
for Sensor:=1 to SensorNum do begin
  GetParamFileLine;
  SensorChannel[Sensor] := WordVal(ParamLine,code);
  GetParamFileLine;
  XSensorPos[Sensor] := WordVal (ParamLine,code);
  YSensorPos[Sensor] :=  WordVal(copy
    (ParamLine,code+1,255),code);
  for Pinhole:=1 to 4 do begin
    GetParamFileLine;
    SensorCalib[Sensor,Pinhole] := RealVal
      (ParamLine,code);
  end;
```

```
  end; {for Sensor}
  GetParamFileLine;
  TempChannel : = WordVal(ParamLine,code);  { assume
only one thermometer at fir
  Get ParamFileLine;
  TempCalPts := WordVal(ParamLine,code);
  for i:=1 to TempCalPts do begin
    { Resistance[i], Temperature[i] may not be integers
}
    GetParamFi  Line;
    Resistance  i] := RealVal(ParamLine,code);
    Temperature[i] :=
RealVal(copy(ParamLine,code+1,255),code) + 273.15;
  end;
  for i:=1 to 3 do begin  {read in Fast, Medium, & Slow
Calib Delay values }
    GetParamFileLine;
    CalibDelays[i,1] := LongVal(ParamLine,code1);
    CalibDelays[i,2] :=
trunc(RealVal(copy(ParamLine,code1
      +1,255),code2)*10+0.5)
    CalibDelays[1,3] :=
trunc(RealVal(copy(Paramline,code2+code1+1, 1,255),
 code) *10
  end;
  close(ParamFile);
  Resistance [0] := Resistance[1] * 2 - Resistance[2];
  Temperature [0] := Tempeature[1] *2 - Temperature[2]
end;

{
  ****************************
  * Option 1: Display Profile*
  ****************************

}
procedure Display1;
```

```
var
  key: Char;
  v,code: Integer;
  Leave,NewSensor: Boolean;
begin
  Sensor := Display1Sensor;
  Display := true;
  Leave := false
  repeat
    SetScreen:
    gotoxy(1,25);
    write('(1) DISPLAY PROFILE: Press ',CalibSelect
      [NumSensors]', to select sensor #, Q to Quit');
    NewSensor := false
    repeat
      Display := true;
      BeamProfileVariables[Sensor].ScanStep := 4;
      GoodRead(1);
      if TotWtAvg=0 then begin
        Center ('ERROR: Sensor #'+IntStr(Sensor)+
          ' was not found: Try (6) Move or
          (7) Search',24);
        Waitkey;
        exit
    end;
    if GoodCount > 0 then begin
      Xsp[Sensor] := Xaverage;
      Ysp[Sensor] := Yaverage;
    end;
    PrintInfo(Sensor);
    if keypressed then begin
      key := ReadKey;
      v := IntVal(key,code);
      if (v > 0) and (v <=NumSensors) then begin
        Sensor := v;
        MoveBeam(trunc(Xsp[Sensor],
          trunc(Ysp[Sensor]));
```

```
        delay(200);
        NewSensor := true;
      end else Leave := true;
    end;
  until NewSensor or Leave;
  until Leave;
  Display1Sensor := Sensor;
end;

{
  **********************
  * Option 2: History *
  **********************
}
procedure History2;
begin
  gotoxy(1,24);
  writeln('(2) History of Beam Power & Profile');
  Center('History File Not Yet Installed',4);
  Center('will track beam power vs. time',5);
  Center('also compare profiles, XY gain & drift',6);
  WaitKey;
end;

{
  ****************************
  * Option 3: Position Check *
  ****************************
}
procedure Ranger (ffield: Integer; position: Real; fch:
  Char));   { Field 1-4 }
var
  ch: Char;
  CurrField, PPos: Integer;
begin
  CurrField := trunc(3*(position+80000.0)/19;
    { 3 character positions per .00
```

```
    PPos := trunc(3*(position+80000.0)-CurrField*19.0);
    ch := fch;
    if LastField[ffield] < 32767 then begin
      { always display first one }
      if CurrField < LastField[ffield] then
        { see if in new field }
        ch := '/'
      else if CurrField > LastField[ffield] then ch := \';
    end;
    LastField[ffield] := CurrField;
    PPos := PPos+1+20*(ffield-1);
    gotoxy(PPos,20);
    write(ch);

    gotoxy(20*ffield,20);
    write(':');
    if PrintOut then begin
      write(PrintDev,spc(PPos),ch.chr(13),spc(20*ffield),
          ':',chr(13);
      if ffield=4 then writeln(PrintDev);
    end;
  end;

  procedure Position3;
  var
    i,x,y,step,dir: Integer;
    temp: String;
    Xoffs,Yoffs,Xgain,Ygain: Real;

    procedure PrinterOff;
    begin
      if printout then begin
        writeIn(PrintDev,TimeStamp);
    { restore 1/6 line spacing,  non-condensed }
      write(PrintDev,chr(27), 'A',chr(12),chr(18));
      write(PrintDev,chr(12)');
      close(PrintDev);
```

```
    PrintOut := false;
  end;
end;

begin
  for i:=1 to 4 do { set up initial fields for Ranger }
    LastField[i] := 32767;
  Display := false;
  repeat
    repeat
    gotoxy(1,21);
    if PositionCheckMode = GainDrift then
      writeln('      X Offset    X Gain    Y Offset
    else
      writeln('     X1           Y1            X2
    if PositionCheckMode = GainDrift then temp :=
      'Gain/Drift Mode'
      else temp := 'Straight X/Y Mode';
    if PositionMethodMode = Fast then temp := temp +
      ' Fast Method'
      else temp := temp + ' Slow Method';
    Center(temp,22);
    writeln;
    write(Blank);
    for Sensor:=1 to 2 do begin
      gotoxy(20,22+Sensor);
      write('Sensor #',Sensor,Power[Sensor]:
        5:1,'mW, Center: ');
      PrintXY(Xsp[Sensor],Ysp[Sensor]);
      write ('     ');
    end; {Sensor}
    Center('(3) POSITION CHECK: P-Printout
      0.9 Speed  Q-Quit', 25);

    for Sensor:=1 to 2 do begin

  (*
```

```
      MoveBeam(32767,32767);  { for Stuart's testing
delay(50);
*)
      if PositionMethodMode = Fast then begin
      MoveBeam(trunc(Xsp{Sensor]),
        trunc(Ysp[Sensor]);
      delay(250);
      end else begin
        x := trunc(Xsp[3-Sensor]);
        y := trunc(Ysp[3-Sensor]);
        step := 64;
        dir := trunc(Xsp[Sensor]) - x;
        if dir > 0 then begin
          while x < Xsp[Sensor] do begin
            MoveBeam(x,y);
            inc(x,step);
          end;
          dec(x,step);
        end else begin
          while x > Xsp[Sensor] do begin
            MoveBeam(x,y);
            dec(x,step);
          end;
          inc(x,step);
        end;
        dir := trunc(Ysp[Sensor]) - y;
        if dir > 0 then begin
          while y < Ysp[Sensor] do begin
            MoveBeam(x,y);
            inc(y,step);
          end;
          dec(y,step);
        end else begin
          while y > Xsp[Sensor] do begin
            MoveBeam(x,y);
            dec(y,step);
          end;
```

```
      inc(y,step);
    end;
    MoveBeam(trunc(Xsp[Sensor]),
      trunc(Ysp[Sensor]));
  end;
  Display := false;
  BeamProfileVariables[Sensor].ScanStep := 4;
  GoodRead(1);
  if GoodCount > 0 then begin
    Xsp[Sensor] := Xaverage;{ update centers }
    Ysp[Sensor] := Yaverage;
  end;
end; {Sensor}
Xoffs := ( Xsp[1]+Xsp[2]-XSensorPos[1]-
  XSensorPos[2] )/2*3.556); {mils}
Yoffs := ( Ysp[1]+Ysp[2]-YSensorPos[1]-
  YSensorPos[2] )/2*3.556);
Xgain := ((Xsp[1]+Xsp[2]/(XSensorPos[1]-
  XSensorPos[2]-1)/*10000; {mils pe
Yg..n := ((Ysp[1]+Ysp[2]/(YSensorPos[1]-
  YSensorPos[2]-1)/*10000;

window(1,1,80,20);
gotoxy(1,20);
writeln;
window(1,1,80,25);
if PositionCheckMode = GainDrift then begin
  Ranger(1,Xoffs,'X');
  Ranger(2,Xgain,'x');
  Ranger(3,Yoffs,'Y');
  Ranger(4,Ygain,'y');
end else begin
  Ranger(1,Xsp[1], 'X');
  Ranger(2,Ysp[1], 'Y');
  Ranger(3,Xsp[2], 'x');
  Ranger(4,Ysp[2], 'y');
end;
```

```
    if not keypressed then delay(PosChkSpeed);
  until keypressed;

  key := upcase(Readkey);
  case key of
    'P': begin
      gotoxy(1,1);
      Center('Do you want a printout (y/n) ?' 25);
      if (upcase(ReadKey) = 'Y') and not PrintOut
        then begin
        PrintOut := true;
        assign(PrintDev, 'PRN');
        rewrite(PrintDev);
        writeln(PrintDev,'X is offset,
          x is length change of 10',
        '" part, both .3 mils / div');
        writeln(PrintDev,spc(85),TimeStamp);
        writeln(PrintDev),
      writeln(PrintDev,chr(27),'A',chr(1),
        chr(15));  { 1/180 LPI, 17 p
      end else PrinterOff;
    end
  '0'..'9': PosChkSpeed := (ord(key)-ord('0')) * 1000
{ms};
  'M' : if PositionMethodMode = Slow then
    PostionMethodMode := Fast
    else PositionMethodMode := Slow;
  'T': if PositionCheckMode = GainDrift
    then PositionCheckMode := StraightXY
    else PositionCheckMode := GainDrift;
  'Q', 'X': begin
      PrinterOff;
      exit;
      end;
    end;
  until false; { loop indefinitely }
end;
```

```
{
  ************************
  * Option 4: Beam Power *
  ************************

}
procedure Power 4;
{
  as fast as possible, sound output
  have dropped tests for 65535,0 on X#, Y#
  rework sounds, can speed with pre-calculation
}
var
  code,ordkey,ADC,col,row,ScanStep,waste: Integer;
  PowerSum : rray[1..3] of LongInt;
  XCorner,Ycorner,TotalWeight: LongInt;
  AvgArray : Array [1..PowerMvAvg] of LongInt;
  AvgIndex,AvgCount : Byte;
label
  Power4a,Power4b,Power4c;
begin
  gotoxy(1,25);
  write('(4) POWER: Press ',CalibSelect[NumSensors],
    'to select sensor #, Q to Quit');
  Sensor := 1;
Power4a:
  AvgCount:=0;
  AvgIndex : = 0;
  PowerSum[Sensor] : = 0;
  ScanStep := 4;  { 1.4 mil = matches that used for
calib }
  Xcorner := trunc(Xsp[Sensor] - ((ColMax+1)/2.0)
    * ScanStep);
  Ycorner := trunc(Ysp[Sensor] - ((RowMax+1)/2.0)
    * ScanStep);
  SetupBeamProfile(BeamProfileVariables,Sensor,-1,-1,-
1,-1);
```

```
  ClearLine(24);
Power4b:
  TotalWeight := 0;
  for col:=1 to ColMax do begin
    Xpos := Xcorner + col * ScanStep;
    Ypos := Ycorner; { extra delay on first }
    MoveBeam(Xpos,Ypos);
    waste:=0
    while waste < BeamProfileVariables[Sensor].
      ColToColDelay do Inc(waste);
    for row:=1 to RowMax do begin  { scream . . . }
      inc(Ypos,ScanStep);
      AxisMove(Ypos,2);  { move laser beam }
      waste:=0
      while waste < BeamProfileVariables[Sensor].
        RowToRowDelay do Inc.(waste);
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
      if TotalWeight + ADC > 32768 then begin
        Center ('Sensor #'+IntStr(Sensor)+ in
         Saturation',24); goto Power4c;
     end;
     TotalWeight := TotalWeight + ADC; { only overflow
       @22*22*60 }
     end; {for row}
     Xpos := Xcorner;
     Ypos := Ycorner;
     MoveBeam(Xpos,Ypos);
   end; {for col}
   Total Weight := TotalWeight - SensorBkgnd[Sensor];
   if AvgIndex=PowerMvAvg then AvgIndex:=1
   else Inc(AvgIndex);
   if AvgCount<PowerMvAvg then Inc(AvgCount)
   else Dec(PowerSum[Sensor],AvgArray[AvgIndex]);
   AvgArray[AvgIndex]:=TotalWeight;
   Inc(PowerSum[Sensor],TotalWeight);
   gotoxy(5,20+Sensor);
   write ('Sensor #',Sensor,(PowerSum[Sensor]/AvgCount)
```

```
    *SensorCalib[Sensor,3]:6:2,
    ' mW Averaged Reading;(',TotalWeight*SensorCalib
      [Sensor,3]:6:2,
    ' mW Unfiltered Reading)');
  if SoundOn then begin
    Sound(1000);
    delay (80);
    if TotalWeight<1 then Sound(37)
    else Sound(trunc(30*ln(TotalWeight)+37));
    delay(200);
  end;
{
  If 10 * round then recheck centers with goodread &
    Xsp[]=XAvg range, duration, need time
}

Power4c:
  if not keypressed then goto Power 4b
  else begin
    ordkey := IntVal(ReadKey,code);
    if (ordkey > 0) and (ordkey <= NumSensors)
      then begin
      Sensor := ordkey;
      goto Power4a;
    end;
  end;
end;

{
    ************************
    *Option 5:  Beam Focus *
    ************************
}
procedure Focus5;
var
  code,ordkey,x,i,i2,stp:  Integer;
  LocRowMax,cwi,rwi,HiPoints,col,row,Try,ADC:  Integer;
```

```
  cwmax,rwmax,ratio,Xaverage,Yaverage:  Real;
  tw,cw,rw,cplo,cphi,rplo,rphi:  Real;
  TotalWeight,Xcorner,Ycorner: LongInt;
  FocusColWeight,FocusRowWeight:  array[1..30] of Real;
label
  Focus5a;
begin
  Sensor := Focus5Sensor;
{
  assumes 22x22 array for now
  1/e^2 but acting on area-integrated power
}
  Setscreen;
Focus5a:
  gotoxy (1,25);
  write('(5) Focus Parameters:  Press' ,CalibSelect
    [NumSensors], 'for sensor #, S to search, Q to
Quit');
  with BeamProfileVariables[Sensor]do begin
    SetupBeamProfile(BeamProfileVariables,Sensor,
      Xsp[Sensor],Ysp[Sensor],
      4{ScanStep},trunc(Bk[Sensor] + 0.5));
    GetBeamProfile(BeamProfileVariables,Sensor);
    if TotalWeight <> 0 then begin
      Xaverage :=Xweight / TotalWeight;
      Yaverage :=Yweight / TotalWeight;
      Xsp[Sensor] := Xaverage;
      Ysp[Sensor] := Yaverage;
    end;
{
  Now assume only ColMax=RowMax (=22)
}
    tw := TotalWeight - SensorBkgnd[Sensor];
    if tw <= 0 then begin
      tw := 1;
      Center('BEAM NOT FOUND',24);
      WaitKey;
```

```
      Display := true;
      BeamFound := FindHoles(Sensor); { scans for
        2-mil hole for that sensor }
      if AbortFlag then exit;
    end;
    delay (1000);
    SetScreen;
    LocRowMax; := ColMax;
    if RowMax > ColMax then LocRowMax := RowMax;

    cwmax := 0;
    if OpMode = TextOp then stp := 2 else stp := 1;
    gotoxy (79,1);
    write ('X');
    for i:=1 to LocRowMax do begin
    cw := ColWeight[i];
    cw := cw - SensorBkgnd[Sensor]/ColMax;
    if cw < 1 then cw := 0.5;
    cw := 6*(ln(cw/tw)+8); { from 2 to 39
     .5/(22*22*220) to 1 }
    if cwmax < cw then cwmax := cw;
    FocusColWeight[i] := cw;
    cwi := trunc(cw*16); { *1.02*lamda# ? }
    if cwi < 1 then cwi := 1
    else if cwi > 79*8 then cwi := 79*8;
    i2 := i*7;
    if i mod stp = 0 then begin
      if OpMode = TextOp then begin
        gotoxy(1,i div stp+1);
        write(many('-',cwi div 8+1),'*');
      end else begin
        Circle(3+cwi,i2,2);
        SetLineStyle(DottedLn,0,NormWidth);
        Line(0,i2,cwi,i2);
      end;
    end;
  end; {for i}
```

```
cwmax := cwmax -6; { to find 1/e^2 of beam }
x := trunc(cwmax * 16);
if x < 1 then x := i;
if x > 79*8 then x := 79*8;
if OpMode = TextOp then begin
  gotoxy(X div 8,12);
  write ('^');
end else Line (x,0,x,160);

gotoxy (79,13);
write ('Y');
rwmax := 0;
for i:= 1 to LocRowMax do begin
  rw := RowWeight[i];
  rw := rw - SensorBkgnd[Sensor]/RowMax;
  if rw < 1 then rw := 0.5;
  rw := 6*(ln(rw/tw)+8; { tw# really not wanted here,
    want absolute }
  if rwmax < rw then rwmax := rw;
  FocusRowWeight[i] := rw;
  rwi := trunc(rw*16); {*1.02*lamda# ? )
  if rw < 1 then rw :=  1;
  if rw > 79*8 then rw := 79*8;
  i2 := 155+i*7+7;
  if i mod stp = 0 then begin
    if OpMode = TextOp then begin
      gottoxy(1,12+i div stp);
      write (many('-',rwi div 8+1), -'
    end else begin
      Circle(3+rwi,i2,2);
      Line (0,i2,rwi,i2);
    end;
  end;
end; {for i}


rwmax :=  rwmax - 6;
```

```
x :=  trunc(rwmax * 16);
if x < 1 then x :=  1;
if x > 79*8 then x :=  79*8;
if OpMode = TextOp then begin
  gotoxy(x div 8,23);
  write ('^');
end else Line (x,165,x,320);
for i:=1 to LocRowMax-1 do begin
  if (FocusColWeight[i+1] > cwmax) and
    (FocusColWeight[i] < cwmax) then
    cplo := i + (cwmax - FocusColWeight[i]) /
      (FocusColWeight[i+1] - FocusColWeight[i]);
  if (FocusColWeight[i+1] < cwmax) and
    (FocusColWeight[i] > cwmax) then
    cphi := i + (cwmax - FocusColWeight[i] /
      (FocusColWeight[i+1] - FocusColWeight[i]);
  if (FocusRowWeight[i+1] > rwmax and
    (FocusRowWeight[i] < rwmax) then
    rplo := i + (rwmax - FocusRowWeight[i] /
      (FocusRowWeight[i+1] - FocusRowWeight[i]);
  if (FocusRowWeight[i+1] < rwmax) and
    (FocusRowWeight[i] > rumax) then
    rphi :=  i + (rwmax - FocusRowWeight[i]) /
      (FocusRowWeight [i+1] - FocusRowWeight[i]);
end;
gotoxy (1,24);
write(Blank);
gotoxy (15,24);
write ('Sensor #',Sensor,' Average Width ');
write ((cphi-cplo+rphi-rplo)*4/(2*3.556):4:1);
if abs(rphi-rplo)<1E-5 then ratio := 0
  else ratio := (cphi-cplo/(rphi-rplo);
write (' mils  X/Y =',ratio:4:2);
```

```
  if not keypressed then goto Focus5a
  else begin
    key :=  upcase(ReadKey);
    if key='S' then begin
    Xsp[Sensor] := Xpos;
    Ysp[Sensor] := Ypos;
    Display :=  true;
    if FindHoles2(Sensor) then begin; { search around
      new center }
      Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
      Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
      Xpos := trunc(Xsp[Sensor]);
      Ypos := trunc(Ysp[Sensor]);
    end;
    goto Focus5a
  end else begin
    ordkey := IntVal(key,code);
    if (ordkey > 0) and (ordkey <= NumSensors) then
      begin
      Sensor := ordkey;
      MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp
        [Sensor]));
      delay (200);
      goto Focus5a
      end;
    end;
  end;
  end;
  Focus5Sensor := Sensor;
end;

{
    ************************
    *Option 6:  Move Beam  *
    ************************
}
procedure Move6;
```

```
var
  KeyVal,code,i,ADC: Integer;
  InputLine,indent: String;
label
  Move6a,Move6b,Move6c;
begin
  Xpos := XRest;
  Ypos := YRest;
  Sensor := 1;
Move6a:
  TextMode (C80);
  gotoxy(1,25);
  write('(6) MOVE BEAM:');
  indent := many (' ', 19);
  gotoxy(1,1);

  writeln(indent,'Sensor  Default  Last Found  Power');
  writeln(indent,         Position Position    (mW)');
  writeln(indent, ______  ________ ___________ _______)
  gotoxy(1,5+NumSensors);
  write(indent,'Beam Rest Position: ');
  PrintXY(XRest,YRest);
  Center('KeyPress Action                          ',17);
  Center('________ ______                          ',18);
  Center(' '+CalibSelect[NumSensors] +
  ' Change sensor #                  ',19);
  Center (' X  Reposition X coord. of beam         ',20);
  Center (' Y  Reposition Y coord. of beam         ',21);
  Center (' S  Search for hole of sensor from
         current position   ',22);
  Center (' Q Quit move Beam function              ',23);
  Center ('*****************************',11);
  Center ('*                           *',12);
  Center ('*                           *',13);
  Center ('*****************************',14);
  for i:=1 to NumSensors do begin
    gotoxy(23,3+;);
```

```
      write(i,'    ');
      PrintXY(XSensorPos[i],YSensorPos[i]);
      write('  ');
      PrintXY(Xsp[i],Ysp[i];
      write ('  ');
      write(Power[i]:5:1);
    end;
  Move6b:
    Clearline(15);
    MoveBeam (Xpos,Ypos);
  Move6c:
    Pause (30000);
    ADC := ReadLaserBoard(SensorChannel[Sensor]);
    gotoxy (28,12);
    write ('Sensor #',Sensor,'Power: ',ADC':3);
    gotoxy(21,13);
    write('Current Beam Position: ');
    PrintXY (Xpos,Ypos);
    if SoundOn and (ADC > Bk[Sensor]) then begin
    Sound(1000);
    delay(10);
    Sound(30*ADC+37);
    delay(20);
  end;
  case upcase(key) of
    ' ': goto Move6c;
    'Q': exit;
    'X': begin
      gotoxy(30,15);
      write('New X Position =');
      readln(InputLine);
      if InputLine<>'' then Xpos := WordVal(InputLine,
        code);
      goto Move6b;
    end;
    'Y': begin
      gotoxy(30,15);
```

```
        write('New Y Position =');
        readln(InputLine);
        if InputLine<>''then Ypos := WordVal(InputLine,
          code);
        goto Move6b;
      end;
      'S': begin
        Xsp[Sensor] :=  Xpos;
        Ysp[Sensor] :=  Ypos;
        SetScreen;
        Display := true;
        If FindHoles2(Sensor) then begin {start again
          at this new center }
          Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
          Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
          Xpos := trunc (Xsp[Sensor]);
          Ypos := trunc (Ysp[Sensor]);
        end;
        if AbortFlag then Waitkey;
        goto Move6a;
      emd;
    else begin
      KeyVal :=  IntVal (key,code);
      if (KeyVal > 0) and KeyVal <= NumSensors) then begin
        Sensor :=  KeyVal;
        go to Move66;
        end;
      end;
    end;
    goto Move6c;
    end;
```

```
{
    ************************
    *Option 7: Find Pinholes*
    ************************
}
procedure Search7;
begin
  SetScreen;
  Sensor := 1;
  Display := true;
  BeamFound := FindHoles(Sensor); {scans for 2-mil hole
    around original center}
  if not AbortFlag then Pause(20000);
  Sensor := 2;
  BeamFound := FindHoles(Sensor);
  if not AbortFlag then Pause(20000);
end;

{
    ****************************
    *Option 8: Calibrate System *
    ****************************
}
procedure Calibr8;
const
  CalibHoleSeparation = 890;
var
  Calibkey: Char;
  ADC,i,code,CalibAttempts: Integer;
  tableX,tableY,FullPart:  Real;
  RowStartXsp,RowStartYsp,oldXsp,oldYsp: Real;
  row,col,Xcount,Ycount: Integer;
  CalibCoords: array[1..50,1..50,1..2] of Real;
  CalibTimeout,GoodRow,GoodCol: Boolean;
  SpiralDelay: Word; { in milliseconds }
  BetweenRowsDelay,BetweenColsDelay: Word;
    {in tenths of seconds }
```

```
  temp,Indent: String;
label
  AbortCalib;

procedure CalibrationDelay(time{in 1/10 secs}:
LongInt);
var
  sec,TotalSec:  LongInt;
begin
  if time > 0 then begin
    ClearLine(24);
    gotoxy(33,24)
    sec := 1;
    TotalSec := time div 10;
    if TotalSec > 0 then write('Waiting ...');
    while (sec <= TotalSec) and no keypressed do begin
      gotoxy(45,24);
      write(TotalSec + 1 - sec,' ');
      delay(999);
      inc(sec);
    end;
    delay(100 * time mod 10);
    while keypressed do key := ReadKey;
    ClearLine(24);
  end;
end;

function StartCalibFile: Boolean;
begin
  StartCalibFile := true;
  assign(CalOutFile,CalibFileName);
  {$I-}
  rewrite(CalOutFile);
  if IOresult<>0 then begin
    Center('Cannot write to Calibration File',24);
    WaitKey;
    StartCalibFile := false;
```

```
      exit;
    end;
    {$I+}
    writeln(CalOutFile,'! Calibration File V' + VersionId
      + ', started ',TimeStamp;
    close(CalOutFile);
  end;

  function UpdateCalibFileRow: Boolean;
  var
    i: Integer;
    FoundBeam1,FoundBeam2: Boolean;
    SensorPosX,SensorPosY: array[1..6] of Real;
  begin
    UpdateCalibFileRow := true;
    assign(CalOutFile,CalibFileName);
    {$I-}
    append(CalOutFile);
    if IOresult<>0 then begin
      Center('Error writing to Calibration File',24);
      WaitKey;
      UpdateCalibFileRow := false;
      exit;
    end;
    {$I+}
    MoveBeam(trunc(Xsp[1]),trunc(Ysp[1]));
    CalibrationDelay(BetweenRowsDelay);
    SetupFindBeam(FindBeamVariables,1{Sensor},-1{Xsp[1]},-
      1{Ysp[1]},40{SpiralSideMax},
      1{SpiralTries},4{ScanStep},trunc(Bk[1]+0.5)
        {BkgndThreshold},
      20{SpiralDelay},1{NumCenterPasses});
    FoundBeam1 :=
  FindBeam(FindBeamVariables,1,NormalFind);
    SensorPosX[1] := FindBeamVariables[1].Xavg;
    SensorPosY[1] := FindBeamVariables[1].Yavg;
    MoveBeam(trunc(Xsp[2]),trunc(Ysp[2]));
```

```
  CalibrationDelay(BetweenRowsDelay);
  SetupFindBeam(FindBeamVariables,2{Sensor},-1{Xsp[2]},-
    1{Ysp[2]},40{SpiralSideMax},
    1{SpiralTries},4{ScanStep},trunc(Bk[1]+0.5)
      {BkgndThreshold},
    20{SpiralDelay},1{NumCenterPasses});
  FoundBeam2 :=
FindBeam(FindBeamVariables,2,NormalFind);
  SensorPosX[2] := FindBeamVariables[2].Xavg;
  SensorPosY[2] := FindBeamVariables[2].Yavg;
  for i:=3 to 6 do begin
    SensorPosX[i] := 0;
    SensorPosY[i] := 0;
  end;
  if not (FoundBeam1 and FoundBeam2) then
    write(CalOutFile,'T ');
  for i:=1 to 6 do begin
    if i > 1 then write(CalOutFile,',');
    write(CalOutFile,RealStr(SensorPosX[i],2),',',
      RealStr(SensorPosY[i],2));
  end;
  writeln(CalOutFile);
  close(CalOutFile);
end;

function UpdateCalibFileCol: Boolean;
var
  Resist,Temper: Real;
begin
  UpdateCalibFileCol := true;
  ReadTemperature(Resist,Temper);
  assign(CalOutFile,CalibFileName);
  {$I-}
  append(CalOutfile);
  if IOresult<>0 then begin
    Center('Error writing to Calibration File',24);
    WaitKey;
```

```
    UpdateCalibFileCol := false;
    exit;
  end;
  {$I+}
  GetTimeStamp;
  if CalibTimeout then write(CalOutFile,'T ');
  writeln(CalOutFile,RealStr(Xsp[Sensor],2),',',RealStr
(Ysp[Sensor],2),',',RealStr(Power[Sensor],2),',',Real
    Str (Temper,2),',',CurrentTime);
  close(CalOutFile);
  CalibTimeout := false;
end;

function QuickCalibSearch: Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor],40{SpiralSideMax},1{SpiralTries},16
     {ScanStep},
trunc(Bk[Sensor]+0.5),SpiralDelay,3{NumCenterPasses});
  if SpiralSearchForHole(FindBeamVariables,Sensor) then
    begin
    QuickCalibSearch := true;
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end else QuickCalibSearch := false;
end;

function CalibFind: Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor],40{SpiralSideMax},1{SpiralTries},
    16{ScanStep},trunc(Bk[Sensor]+0.5),SpiralDelay,
    2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind) then
    begin
    CalibFind := true;
```

```
    Xsp[Sensor] := FindBeamVariables[Sensor].Xavg;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yavg;
  end else CalibFind := false;
end;

begin
  Calibrate := true;
  Center('Beam Calibration Function',1);
  gotoxy(1,8);
  Indent := many(' ',16);
  writeln(Indent,'Type        Fast     Medium     Slow');
  writeln(Indent,'--------  ------    ------    -----');
  writeln(Indent,'Spiral
',CalibDelays[1,1]:4,';
    CalibDelays[2,1]:4,'    ',CalibDelays[3,1]:4,'
ms');
  writeln(Indent,'Between Points ',CalibDelays[1,2]/
   10:5:1,'   ';
    CalibDelays[2,2]/10:5:1,'
      ',CalibDelays[3,2]/10:5:1, ' secs');
  writeln(Indent,'BetweenRows
',CalibDelays[1,3]/10:5:1, CalibDelays[2,3]/10:5:1,'
  ',CalibDelays[3,3]'/'10:5:1,
    ' secs';);
  writeln(Indent,'     _____     _____     _____');
  writeln(Indent,'TOTAL     ',
    (2.5*10*49*49+CalibDelays[1,2]*49*49+CalibDelays[1,3
      ] *49*3)/36000:5:1,'
    (2.5*10*49*49+CalibDelays[2,2]*49*49+CalibDelays[2,3
      ] *49*3)/36000:5:1,'
    (2.5*10*49*49+CalibDelays[3,2]*49*49+CalibDelays[3,3
      ] *49*3)/36000:5:1,'   hou
  writeln;
  Center('Fast, Medium, Slow or Different
 Calibration?',18);
  CalibKey := upcase(ReadKey);
  case CalibKey of
```

```
    'F': begin
        SpiralDelay     := CalibDelays[1,1]; { in
milliseconds }
        BetweenColsDelay:= CalibDelays[1,2]; { in tenths
of seconds }
        BetweenRowsDelay:= CalibDelays[1,3];
      end;
    'M': begin
        SpiralDelay     := CalibDelays[2,1];
        BetweenColsDelay:= CalibDelays[2,2];
        BetweenRowsDelay:= CalibDelays[2,3];
      end;
    'S': begin
        SpiralDelay     := CalibDelays[3,1];
        BetweenColsDelay:= CalibDelays[3,2];
        BetweenRowsDelay:= CalibDelays[3,3];
      end;
    'D': begin
        writeln;
        writeln;
        write('Enter Spiral Delay (in ms): ');
        readln(temp);
        if temp='' then exit
          else SpiralDelay := LongVal(temp,code);
        write('Enter Between Points Delay (in secs): ');
        readln(temp);
        if temp = '' then exit
          else BetweenColsDelay :=
            trunc(RealVal(temp,code) *10+0.5);
        write('Enter Between Rows Delay (in secs): ');
        readln(temp);
        if temp = '' then exit
          else BetweenRowsDelay :=
            trunc(RealVal(temp,code) *10+0.5);
        end;
    else exit;
end;
```

```
SetScreen;
case CalibKey of
  'F': Center('Fast Beam Calibration, started
'+TimeStamp,1);
  'M': Center('Medium Beam Calibration, started
    '+TimeStamp,1);
  'S': Center('Slow Beam Calibration, started
'+TimeStamp,1);
  'D': Center('Different Beam Calibration, started
    '+TimeStamp,1);
end;

{
  Search for corner of calibration box:

  1.  move out Y-axis until edge is reached
      edge is qualified by two consecutive hole scans
      that fail
  2.  move out X-axis until edge is reached
      this places laser beam as close as possible to
      (0,0)

  If a hole is found on the same axis after one has not
  been found, alg assumes beam was somehow blocked over
  the first hole and aborts.  If two consecutive holes
  are blocked, the alg will think it has found the edge
  of the box and start calibrating.

  Display := false;
  Sensor := 3;
  SetupFindBeam(FindBeamVariables,Sensor,32767,32767,
    75{SpiralSideMax},1{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor]+0.5),SpiralDelay,0{NumCenterPasses}
    );
  if not FindBeam(FindBeamVariables,Sensor,NormalFind)
    then begin
    beep;
```

```
  Center('Can''t detect calibration plate!',24);
  WaitKey;
  exit;
end;
Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
Ycount := -1;
repeat
  inc(Ycount);
  oldXsp := Xsp[Sensor];
  oldYsp := Ysp[Sensor];
  Ysp[Sensor] := Ysp[Sensor] - CalibHoleSeparation;
until not QuickCalibSearch;
if AbortFlag then exit;
Xsp[Sensor] := oldXsp;
Ysp[Sensor] := oldYsp - 2 * CalibHoleSeparation;
if QuickCalibSearch then begin
  beep;
  Center('Blocked hole while scanning for edge.
    Aborting.',24);
  WaitKey;
  exit;
end;
if AbortFlag then exit;
Xsp[Sensor] := oldXsp;
Ysp[Sensor] := oldYsp;
Xcount := -1;
repeat
  inc(Xcount);
  oldXsp := Xsp[Sensor];
  oldYsp := Ysp[Sensor];
  Xsp[Sensor] := Xsp[Sensor] - CalibHoleSeparation;
until not QuickCalibSearch;
if AbortFlag then exit;
Xsp[Sensor] := oldXsp - 2 * CalibHoleSeparation;
Ysp[Sensor] := oldYsp;
if QuickCalibSearch then begin
```

```
      beep;
      Center('Blocked hole while scanning for edge.
        Aborting.',24);
      WaitKey;
      exit;
    end;
    if AbortFlag then exit;
    Xsp[Sensor] := oldXsp;
    Ysp[Sensor] := oldYsp;
    MoveBeam(trunc(oldXsp),trunc(oldYsp));

  {
    Calibration

    Alg moves out each row (Y-axis), then out each column
    (X-axis) looking for a hole.  The current alg looks
    for a set of 49 x 49 holes.  For each position,
    whether it finds a hole or not, it makes an entry in
    the CALIB.DAT file.  Entries in this file for holes
    for which the laser beam wasn't found after 10
    attempts are preceded by a 'T'.
  }
    if not StartCalibFile then exit;
    CalibTimeout := false;
    for row := 1 to 49 do begin
      GoodRow := CalibFind;
      if not UpdateCalibFileRow then exit;
      MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
      CalibrationDelay(BetweenRowsDelay);
      for col := 1 to 49 do begin
        if AbortFlag then exit;
        oldXsp := Xsp[Sensor];
        oldYsp := Ysp[Sensor];
        CalibAttempts := 0;
        repeat
          GoodCol := CalibFind;
          if AbortFlag then exit;
```

```
          if not GoodCol then begin
            inc(CalibAttempts);
            beep;
            Center('Calibration hole ('+IntStr(row)+','+
              IntStr(col)+') not found. Dust?',24);
            if CalibAttempts < 10 then GoodCol := false
              else begin
                GoodCol := true;{ go on after 10 attempts }
                CalibTimeout := true; { set calibration
                                        timeout flag }
              end;
            Xsp[Sensor] := oldXsp;
            Ysp[Sensor] := oldYsp;
          end;
          if GoodCol then begin
            if col = 1 then begin
              RowStartXsp := Xsp[Sensor];
              RowStartYsp := Ysp[Sensor];
            end;
            Ysp[Sensor] := Ysp[Sensor] +
CalibHoleSeparation;
          end;
        until GoodCol;
        if not UpdateCalibFileCol then exit;
        Center('Calibration hole
          ('+IntStr(row)+','+IntStr( col),24);
        if col < 49 then begin
          MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
          CalibrationDelay(BetweenColsDelay);
        end;
      end; {col}
      Xsp[Sensor] := RowStartXsp;
      if row < 49 then begin
        oldXsp := Xsp[Sensor];
        oldYsp := Ysp[Sensor];
        CalibAttempts := 0;
        repeat
```

```
          Xsp[Sensor] := Xsp[Sensor] +
CalibHoleSeparation;
          Ysp[Sensor] := RowStartYsp;
          GoodRow := CalibFind;
          if AbortFlag then exit;
          if not GoodRow then begin
            inc(CalibAttempts);
            beep;
            Center('Can''t locate row '+IntStr(row)+'.
              Dust?',24);
            if CalibAttempts < 10 then GoodCol := false
              else begin
                GoodRow := true;{ go on after 10 attempts }
                CalibTimeout := true; { set calibration
                                        timeout flag }
              end;
            Xsp[Sensor] := oldXsp;
            Ysp[Sensor] := oldYsp;
          end;
        until GoodRow;
      end;
    end; {row}
    if not UpdateCalibFileRow then exit;
    Center('Calibration completed.',24);
    WaitKey;
    exit;

  {*
      --- OLD CALIBRATION ROUTINE ---
      never tested after converted to Pascal
      need revision if its to be used

    MoveBeam(Xpos,Ypos);
    Center('Press a key when Sensor #3 is centered',24);
    gotoxy(28,25);
    write('Sensor #3 reading is');
    repeat
```

```
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
      gotoxy(49,25);
      write(ADC);
      if SoundOn and (ADC > 7) then begin
        Sound(1000);
        delay(10);
        Sound(30*ADC+37);
        delay(20);
        NoSound;
      end;
      delay(300);
    until keypressed;
    while keypressed do key := ReadKey;
    ClearLine(24);
    ClearLine(25);

    if AbortFlag then goto AbortCalib;
    Center('Do you want to run a Full or Partial
      Calibration (F/P)?',24);
    key := upcase(ReadKey);
    if key='F' then FullPart := 0.25
      else FullPart := 10;
    for Sensor:=1 to 2 do begin
      { SlowCalibFind(Sensor); }
      if AbortFlag then goto AbortCalib;
    end;

    tableX := 5;
    repeat
      Sensor := 3;
      Xsp[Sensor] := 32767.0-(0.9539)*3556.0*TableX;
      Ysp[Sensor] := 32767.0+(0.9760)*3556.0*(-5); {
  TableY }
      { here adjust for geometry = or we'll miss holes
        must set parameters before findholes }
      Xpos := trunc(Xsp[Sensor]);   { best guess, improve
                                    with pre-calib }
```

```
      Ypos := trunc(Ysp[Sensor]);
      MoveBeam(Xpos,Ypos);
      CalibrationDelay(20);
      Xpos := 32767;
      Ypos := 32767;
      { FindHole(Sensor); }
      tableY := -5;
      repeat
        Sensor := 3;
        Xsp[Sensor] := 32767.0 - 0.9539 * 3556.0 * tableX;
        Ysp[Sensor] := 32767.0 + 0.9760 * 3556.0 * tableY;
        { Adjust for Geometry }
        XTablePos := trunc(100000*tableX);
        YTablePos := trunc(100000*tableY);
        MoveTable;
        Center('X Table = '+RealStr(tableX,5)+' Y Table =
          '+RealStr(tableY,5),23)
        { CalibFind(Sensor); }
        PrintInfo(Sensor);
        if AbortFlag then goto AbortCalib; { needs
revision? }
        tableY := tableY + FullPart; { .25 or .5 }
      until tableY > 5;
      for Sensor:=1 to 2 do begin { could also use
                                    Position3 routine }
        { SlowCalibFind(Sensor); }
        if AbortFlag then goto AbortCalib; { needs
revision? }
      end;
      tableX := tableX - FullPart;
    until tableX < -5;
    Center('Beam Calibration complete at '+TimeStamp,24);
    XTablePos := 0;
    YTablePos := 0;
    MoveTable;
    WaitKey;
    exit;
```

```
  --- END OF OLD CALIBRATION ROUTINE ---

*)
AbortCalib:
  Center('**** Aborting Beam Calibration ****',24);
  WaitKey;
end;
{-------------------------------------------------------}

procedure ShowTitle;
begin
  TextMode(C80);
  SingleBar := many(#196,26);
  centerln('BEAM, Laser Beam Analysis Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  centerln(SingleBar);
end;

{
  Display Beam's Main Menu
}
procedure DisplayMenu;
var
  i: Integer;
begin
  Calibrate := false;
  ShowTitle;
  window(25,7,80,18);
  gotoxy(1,1);
  writeln('1. Display Profile');
  writeln('2. Beam Power/Profile History');
  writeln('3. Position Check');
  writeln('4. Beam Power');
  writeln('5. Beam Focus');
  writeln('6. Move Beam');
```

```
  writeln('7. Find Pinholes');
  writeln('8. Calibrate System');
  writeln('T. Toggle between Screen Modes');
  writeln('X. Exit - Return to Main Menu');
  window(1,1,80,25);
  gotoxy(29,18);
  if OpMode = TextOp then write(' Text Display Mode ')
    else write('Graphics Display Mode');
  gotoxy(14,20);
  write('Sensor # Default Location    Current
Power(mW)');
  for i:=1 to 3 do begin
    gotoxy(17,20+i);
    write(i);
    gotoxy(25,20+i);
    PrintXY(XSensorPos[i],YSensorPos[i]);
    gotoxy(43,20+i);
    PrintXY(Xsp[i],Ysp[i]);
    gotoxy(59,20+i);
    write(Power[i]:5:1);
  end;
end;

{
  print option line and get selection
}
procedure GetSelection;
var
  SelectionMade: Boolean;
  Resist,Temper: Real;
  code: Integer;
begin
  ReadTemperature(Resist,Temper);
  gotoxy(1,25);
  write('MENU: SELECT OPTION');
  gotoxy(66,25);
  write('Temp: ');
```

```
  if Temper > 70 then begin
    write(' > 70 C');
    beep;
  end else write(Temper:5:1,'C');
  SelectionMade := false;
  repeat
    key := upcase(ReadKey);
    if key in ['1'..'9','T','X'] then begin
      SelectionMade := true;
      if key='T' then if OpMode = TextOp then OpMode :=
        GraphicsOp
        else OpMode := TextOp;
      if key='X' then Select := 9
        else Select := IntVal(key,code);
    end;
  until SelectionMade;
end;

procedure BeamAnalysis;
var
  i: Integer;
begin
  IOSetup;
  for i:=1 to 3 do begin
    Xsp[i] := XSensorPos[i];
    Ysp[i] := YSensorPos[i];
  end;
{
  Main Program Loop
}
  repeat
    Xpos := XRest; { Move beam to rest position }
    Ypos := YRest;
    MoveBeam(Xpos,Ypos);
    DisplayMenu;
    GetSelection;
    ClrScr;
```

```
      case Select of
        1: Display1;
        2: History2;
        3: Position3;
        4: Power4;
        5: Focus5;
        6: Move6;
        7: Search7;
        8: Calibr8;
        9: halt(0);
      end;
    until false;
end;

{ - - - - M A I N   P R O G R A M - - - - -
}

begin  { Beam }

  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);
    { clear variables }
  OldExitProc := ExitProc;
  ExitProc := @ExitBeam;
  CheckBreak := true;
  ShowTitle;
  OpMode := GraphicsOp;
  Display1Sensor := 1;
  Focus5Sensor := 1;
  PositionCheckMode := GainDrift;
  PositionMethodMode := Slow;

  writeln;
  writeln;
  OutFileName := '';
  ReadBeamParamsFile;
  ReadCommandLine(CmdLine);
  CheckForOptions(CmdLine);
```

```
  BeamAnalysis;
  QuietSystemError('Success.');
end.
```

BEAMGR.PCT

```
{-------------------------------------------------------
                General Graphics Routines
-------------------------------------------------------}
procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn, 0, ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x+TextWidth(Text),y+TextHeight(Text));
  SetColor(color);
  OutTextXY(x,y,Text);
end;

function GetInput(maxlen: Integer): String;
var
  key: Char;
  temp: String;
  input: Boolean;
  baseX,baseY,index: Integer;
begin
  temp := '';
  index := 0;
  baseX := getX;
  baseY := getY;
```

```
  key := chr(0);
  while key<>Cr do begin
    MoveTo(baseX+index*8,baseY);
    SetColor(3);
    OutText(chr(219));
    key := ReadKey;
    if key = Ctr1C then Abort;
    MoveTo(baseX+index*8,baseY);
    SetColor(0);
    OutText(chr(219));
    MoveTo(baseX+index*8,baseY);
    SetColor(3);
    if key in [Bs,Cr,' '..'~'] then begin
      if key=Bs then begin
        if index>0 then dec(index);
      end else if key in[' '..'~'] then begin
        OutText(key);
        if index<maxlen then inc(index);
        temp[index] := key;
      end;
    end;
  end;
  temp[0] := chr(index);
  GetInput := temp;
  SetColor(3);
end;

procedure GrAbort(Msg: string);
begin
  Writeln(Msg,': ',GraphErrorMsg(GraphResult));
  Halt(1);
end;

procedure InstallGraphics;
begin
  { Register all the drivers }
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
```

```
    GrAbort('EGA/VGA');

  GraphDriver := Detect;                { autodetect the
                                          hardware }
  InitGraph(GraphDriver,GraphMode,'');  { activate
                                          graphics }
  if GraphResult<>grOk then             { any errors? }
                                          begin
    Writeln('Graphics init error: ', GraphErrorMsg
     (GraphDriver));
    Halt(1);
  end;
  SetVisualPage(0);
  SetActivePage(0);
  ClearDevice;
end;

procedure SetScreen;
begin
  if OpMode = GraphicsOp then begin
    if not GrInstalled then begin
      InstallGraphics;
      GrInstalled := true;
    end else SetGraphMode(GraphMode);
    SetTextStyle(DefaultFont,HorizDir,1);
    SetColor(White);
    SetActivePage(0);
    SetVisualPage(0);
    DirectVideo := false;
  end else begin
    TextMode(C80);
    DirectVideo := true;
  end;
end;

{
  Print action information
```

```
}
procedure Action(Text: String);
var
  i: Integer;
begin
  if OpMode=TextOp then write(Text,' ')
  else begin
    MoveTo(40*8,7*14);
    SetColor(2);
    OverText(Text,20);
    SetColor(3);
  end;
end;

procedure Message(Text: String);
begin
  MoveTo(3*8,24*14-2);
  OverText(Text,length(Text)+5);
end;
^Z
```

```
BEAMIO.INC
{-------------------------------------------------------
                BUILD Miscellaneous I/O Routines
-------------------------------------------------------}

procedure Abort;
var
  ch: Char;
begin
  TextMode(C80);
  ExitBeam;
end;

function FnAbort: Boolean;
var
  ch: Char;
begin
  AbortFlag := false;
  if keypressed then begin
    Center('Do you really want to abort
     (Y or N)?',25);
    repeat
      ch := upcase(ReadKey);
    until ch in ['Y','N'];
    if ch = 'Y' then AbortFlag := true
      else AbortFlag := false;
    gotoxy(1,25);
    write(Blank);
  end;
  FnAbort := AbortFlag;
end;

procedure JbError;
begin
  Center('JB Board I/O Timeout Error. Aborting
     Beam Program',24);
  WaitKey;
```

```
  ClrScr;
end;

procedure SetScreen;
     begin
          if OpMode = GraphicsOp then begin
               if not GrInstalled then begin
                         InstallGraphics;
                         GrInstalled := true;
               end else SetGraphMode(Graph Mode),
               Set TextStyle (DefaultFont, HorizDir,1);
               Set Color(White);
               Set ActivePage(0);
               Set VisualPage(0);
               DirectVideo := false;
          end else begin
               TextMode (c80);
               DirectVideo := true;
          end;
          end;
{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
```

```
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work
     on standard PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;     { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;     { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;     { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;     { get day }
            Day := port[$71];
            port[$70] := $0A;
```

```
                if (port[$71] and $80)=0 then begin
                  port[$70] := $08;     { get month }
                  Mon := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then begin
                    port[$70] := $09;     { get year }
                    Year := port[$71];
                    port[$70] := $0A;
                    if (port[$71] and $80)=0 then good :=
true;
                  end;
                end;
              end;
            end;
          end;
        end;
      end;
    Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
    Minu := (Minu shr 4) * 10 + (Minu and $0F);
    Hour := (Hour shr 4) * 10 + (Hour and $0F);
    Day  := (Day  shr 4) * 10 + (Day  and $0F);
    Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
    Year := (Year shr 4) * 10 + (Year and $0F);
end;

{
  Get a time stamp consisting of the relative day
     and the current time
}
procedure GetTimeStamp;
var
  tday,tmon,tyear,tsec,tminu,thour: str10;
  AmPm: Char;
begin
  GetTime;
  tmon  := PadSp(Mon);
```

```
  tday  := LZero(Day);
  tyear := LZero(Year);
  CurrentDate := tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

function TimeStamp: String;
begin
  GetTimeStamp;
  TimeStamp := CurrentDate + ' ' + CurrentTime;
end;

function ReadLaserBoard(Channel: Integer): Integer;
var
  count: LongInt;
  ADC,i: Integer;
begin
  count := Timeout;
  port[LaserAddr+12] := Channel;
  { must wait 10 us for EOS to go LO (286 @ 4us/loop) }
  i := 1;
  while i < 10 do inc(i);
```

```
  repeat   { then approx 110 us for conversion with
     signal D7 HI }
    dec(count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if PeakReading < ADC then PeakReading := ADC;
  if count = 0 then JbError;
  ReadLaserBoard := ADC;
end;
procedure ReadTemperature(var Resist,Temper: Real);
var
  temp: Real;
  It,ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board
  use 1/T, LOG(R) function TO linearize the
interpolation
}
  ADC := ReadLaserBoard(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC);    { from
20K resistor on JB Board }
  if ADC>214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] > Resist)
do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It]));
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) - ln(Resist)));
  end;
```

```
  Temper := Temper - 273.15;
end;^Z
```

```
BEAMSCAN.INC
{-------------------------------------------------------
                    BEAM Scanning Routines
-------------------------------------------------------
                          ************
                          * MoveBeam *
                          ************

  Routine to move mirrors to X,Y directly
  Note: X,Y can be any kind of variable, we change to
integers
        and get ADC% (for speed: later fixed addrs
routines for each)
}
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 then pos := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
 { x is bit 0, y is bit 1 }
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInXAxis(x: LongInt);
begin
     AxisMove(x,1);
end;

procedure MoveInYAxis(y: LongInt);
begin
     AxisMove(x,1);
end;

begin
     if debug then writeln ('MoveBeam: moving beam to
      ',x',',y);
```

```
  MoveInXAxis(x);
  MoveInYAxis(y);
end;

{
                        ******************
                        * GetBeamProfile *
                        ******************
}
procedure SetupBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer;X,Y: Real;_ScanStep,_BkgndThreshold:
     Integer);
begin
  with BeamProfileVars[Sensor] do begin
    if X                >= 0 then Xpos           :=
trunc(X);
    if Y                >= 0 then Ypos           :=
trunc(Y);
    if _ScanStep        >= 0 then ScanStep       :=
_ScanStep;
    if _BkgndThreshold >= 0 then BkgndThreshold :=
_BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;

procedure GetBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer);
var
  Pass,row,col,ADC,delay,waste: Integer;
  Xcorner,Ycorner: LongInt;
begin
  with BeamProfileVars[Sensor] do begin
```

```
    XYweight        := 0;
    Xweight         := 0;
    Yweight         := 0;
    TotalWeight     := 0;   { for power, not removing
noise till end }
    SaturationCount := 0;
    HiPoints        := 0;
    BeamFound       := false;
    for col:=1 to ColMax do ColWeight[col] := 0;
    for row:=1 to RowMax do RowWeight[row] := 0;
    Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
    Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
    MoveBeam(Xcorner,Ycorner);
    waste := 0;
    while waste < InitialDelay do inc(waste);
    PeakReading := 0;
    for col:=1 to ColMax do begin   { prefer to return
          around edges after line }
      Xpos := Xcorner + col * ScanStep;
      Ypos := Ycorner;
      MoveBeam(Xpos,Ypos);
      waste := 0;
      while waste < ColToColDelay do inc(waste);
      for row:=1 to RowMax do begin
        inc(Ypos,ScanStep);
        MoveInYAxis(Ypos);
        waste := 0;
        while waste < RowToRowDelay do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC < BkgndThreshold then ADC := 0
          else begin
            inc(HiPoints);
            BeamFound := true;
          end;
        if ADC > 210 then inc(SaturationCount);
        inc(TotalWeight,ADC);
        inc(ColWeight[col],ADC);
```

```
          inc(RowWeight[row],ADC);
          BeamMap[row,col] := ADC;
          inc(XYweight,ADC);
          inc(Xweight,Xpos * ADC);   { want floating point
                      numbers here }
          inc(Yweight,Ypos * ADC);
        end; {for row}
      end; {for col}
      MaxSensorReading := PeakReading;
    end;
  end;

  {
                      **********************
                      * DisplayBeamProfile *
                      **********************
  }
  procedure DisplayBeamProfile(var BeamProfileVars:
  BeamProfileVarType;
    Sensor: Integer);
  var
    row,col,gx,gy,a,ADC,cc,cd: Integer;
  begin
    if Display then begin
      gotoxy(70,1);
      write('Sensor #',Sensor);
      if BeamFound then begin
        Center('Found Sensor #'+IntStr(Sensor),25);
        ClearCol(ColMax);
        for col:=1 to ColMax do begin   { prefer to return
                  around edges after line }
          ClearCol(ColMax-col);
          for row:=1 to RowMax do begin
            gx := ((ColMax - col) * 3 + 1) * 8;
            gy := (RowMax - row + 1) * 14;
            ADC :=
  BeamProfileVars[Sensor].BeamMap[row,col];
```

```
            a  := (ADC - trunc(Bk[Sensor]-3)) div
                                DiscreteBeam + 1;
            if a > 15 then a := 15;
            if OpMode = TextOp then begin
              gotoxy((ColMax - col) * 3 + 1,RowMax - row
                                    + 1);
              if ADC < 1 then write('  +') else
write(ADC:3);
            end else begin
              if a < 2 then PutPixel(gx,gy,3)
              else Circle(gx,gy,a);
            end;
            if keypressed then exit;
          end; {for row}
        end; {for col}
      end else Center('Sensor #'+IntStr(Sensor)+' not
found',25);
    end; {if Display}
end;

{
                        ************
                        * GoodRead *
                        ************
}

procedure GoodRead(iter: Integer);
var
  Try: Integer;
begin
  with BeamProfileVariables[Sensor] do begin
    Xaverage  := 0;
    Yaverage  := 0;
    TotWtAvg  := 0;
    GoodCount := 0;
    for Try := 1 to iter do begin
```

```
      SetupBeamProfile(BeamProfileVariables,Sensor,
       Xsp[Sensor],
        Ysp[Sensor],-1{ScanStep},-1{BkgndThreshold});
      GetBeamProfile(BeamProfileVariables,Sensor);
      if Display then
DisplayBeamProfile(BeamProfileVariables,Sensor);
      if XYweight > 0 then begin
        Xaverage := Xweight / XYweight + Xaverage;
        Yaverage := Yweight / XYweight + Yaverage;
        TotWtAvg := TotWtAvg + TotalWeight -
SensorBkgnd[Sensor];
        inc(GoodCount);
      end;
    end; {for Try}
    if GoodCount > 0 then begin
      Xaverage := Xaverage / GoodCount;
      Yaverage := Yaverage / GoodCount;
      TotWtAvg := TotWtAvg / GoodCount;
    end;
    Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
  end;
end;

{-------------------------------------------------------
--
                        ***********
                        * IOSetup *
                        ***********
}
procedure IOSetup;
begin
  Center('--- Hardware Reset ---',23);
  { control byte of Laser Board; A,B out, C 1/2 in 1/2
out and Thermistor Board same }
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Ypos := YRest;
```

```
  MoveBeam(Xpos,Ypos);
end;

{
                        ************
                        * FindBeam *
                        ************
}

procedure SetupFindBeam(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer;X,Y:
Real;_SpiralSideMax,_SpiralTries,_ScanStep,
  _BkgndThreshold,_SpiralDelay,_NumCenterPasses:
Integer);
begin
  with FindBeamVars[Sensor] do begin
    if X                >= 0 then Xval            :=
trunc(X);
    if Y                >= 0 then Yval            :=
trunc(Y);
    if _SpiralSideMax   >= 0 then SpiralSideMax   :=
_SpiralSideMax;
    if _SpiralTries     >= 0 then SpiralTries     :=
_SpiralTries;
    if _ScanStep        >= 0 then ScanStep        :=
_ScanStep;
    if _SpiralDelay     >= 0 then SpiralDelay     :=
_SpiralDelay;
    if _BkgndThreshold  >= 0 then BkgndThreshold  :=
_BkgndThreshold;
    if _NumCenterPasses >= 0 then NumCenterPasses :=
_NumCenterPasses;
  end;
end;
```

```
function SpiralSearchForHole(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer): Boolean;
var
  i,ADC,SpiralSide,waste: Integer;
  X,Y: LongInt;
begin
  with FindBeamVars[Sensor] do begin
    X := trunc(Xval);
    Y := trunc(Yval);
    MoveBeam(X,Y);
    delay(SpiralDelay);
    BeamFound := false;
    SpiralSide := 1;
    while (SpiralSide <= SpiralSideMax) and not
BeamFound do begin
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInXAxis(X);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(X,ScanStep);
        inc(i);
      end;
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInYAxis(Y);
        waste := 0;
        while  aste < SpiralDelay) do inc(waste);
        ADC := eadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(Y,ScanStep);
        inc(i);
      end;
      if FnAbort then exit;
```

```
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := BeamFound;
  end;
end;

function FindBeam(var FindBeamVars:
FindBeamVarType;Sensor: Integer;
  FindMethod: FindMethodType): Boolean;
const
  HoleSeparation = 600;
var
  DisplayFlag: Boolean;
  i,ADC,pass,CantFind,SpiralSide: Integer;
  XspAv,YspAv: Real;
  X,Y,GoodX,GoodY: LongInt;
  BeamFound: Boolean;
begin
  FindBeam := true;
  DisplayFlag := Display;
  with FindBeamVars[Sensor] do begin
    CantFind  := 0;
    repeat
      X := trunc(Xval);
      Y := trunc(Yval);
      MoveBeam(X,Y);   { gets us to best starting
position }
      FlushKey;
      Center('Searching for Sensor #'+IntStr(Sensor)+':
press any key to abort',25);
{
```

```
  now spiral to search for holes
}

SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1{SpiralSide
Max},

-1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},-1{Spira
lDelay},
        -1{NumCenterPasses});
      BeamFound :=
SpiralSearchForHole(FindBeamVariables,Sensor);
      if AbortFlag then exit;
      if not BeamFound then begin
        Center('Can''t find Sensor #'+IntStr(Sensor)+'.
Is beam blocked?',24);

SetupFindBeam(FindBeamVariables,Sensor,X,Y,SpiralSideMax
* 2,
          -1{SpiralTries},-1{ScanStep},BkgndThreshold -
1,-1{SpiralDelay},
          -1{NumCenterPasses});
        inc(CantFind);
      end;
    until (CantFind >= SpiralTries) or BeamFound;
    if CantFind >= SpiralTries then begin
      MoveBeam(Xval,Yval);
      beep;
      Center('Could not find Sensor
#'+IntStr(Sensor),24);
      if not Calibrate then WaitKey;
      FindBeam := false;
      exit;
    end;
    X := Xval;
    Y := Yval;
{
  Verify that we have 2 mil hole...
```

```
  1/2 mil     1 mil ) left rear                2 mil      4
mil ) right front
                    ) of vat
  ) of vat
   4  mil     2 mil )                          1 mil     1/2
mil )

     Sensor #1                                    Sensor #2

   We have the position of one of the four holes: 1/2,
1, 2, or 4 mils.
   We want to find the 2 mil hole closest to the vat.
Note that the orientation of the holes is different for
Sensors 1 and 2.

   Hole Size Check Procedure:

   1.  We may have the position for the 1/2 or 1 mil
hole.  Move towards the vat in the X-axis and search for
the 1 or 2 mil hole.  If the beam is seen, adopt the new
coordinates.

   2.  Now we may have the position for the 4 mil hole.
Move towards the vat in the Y-axis and search for the 2
mil hole.  If the beam is seen, use the new coordinates.
}
    if (Sensor < 3) and (FindMethod = Verify2MilHole)
then begin GoodX := X;
      GoodY := Y;
      if X < 32767 then inc(X,HoleSeparation)
        else dec(X,HoleSeparation);

SetupFindBeam(FindBeamVariables,Sensor,X,Y,SpiralSideMax
div 2,

-1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},-1{Spira
lDelay},
```

```
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
          GoodX := Xval;
          GoodY := Yval;
        end;
      X := GoodX;
      Y := GoodY;
      if AbortFlag then exit;
      if Y < 32767 then inc(Y,HoleSeparation)
        else dec(Y,HoleSeparation);

SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1{SpiralSide
Max},

-1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},-1{Spira
lDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
      X := GoodX;
      Y := GoodY;
      if AbortFlag then exit;
    end;

    if DisplayFlag then begin
      SetScreen;
      gotoxy(70,1);
      write('Sensor #',Sensor);
      if BeamFound then Center('Found Sensor
#'+IntStr(Sensor),25)
      else Center('Sensor #'+IntStr(Sensor)+' not
found',25);
    end;
```

```
    XspAv := 0;
    YspAv := 0;
    if BeamFound then begin
      for pass:=1 to NumCenterPasses do with
BeamProfileVariables[Sensor] do begin
        CantFind := 0;
        { discard the first }

SetupBeamProfile(BeamProfileVariables,Sensor,X,Y,
          4{ScanStep},trunc(Bk[Sensor] + 0.5));
        repeat
          GetBeamProfile(BeamProfileVariables,Sensor);
          if XYweight = 0 then inc(CantFind);
        until (XYweight > 0) or (CantFind > 3);
        if CantFind > 3 then begin
          beep;
          Center('Can''t profile Sensor
#'+IntStr(Sensor)+'. Try (6) Move',24);
          if not Calibrate then WaitKey;
          FindBeam := false;
          exit;
        end;
        if SaturationCount > 0 then begin
          gotoxy(1,24);
          Center('WARNING: '+IntStr(SaturationCount)
            + ' sensor readings near saturation',24);
          delay(1000);
        end;
{
  Later select sensor hole size automatically
}
        X := trunc(Xweight / XYweight);   { better
coordinates }
        Y := trunc(Yweight / XYweight);
        Power[Sensor] := (TotalWeight -
SensorBkgnd[Sensor])
          * (SensorCalib[Sensor,3]);
```

```
          XspAv := XspAv + Xweight / XYweight;
          YspAv := YspAv + Yweight / XYweight;
        end; {for pass}
        if NumCenterPasses > 1 then begin
          XspAv := XspAv / NumCenterPasses;
          YspAv := YspAv / NumCenterPasses;
        end;
      end; {if BeamFound}
      Xval := X;       { update integer coordinates }
      Yval := Y;
      Xavg := XspAv;   { update average coordinates }
      Yavg := YspAv;
      if DisplayFlag then begin
        DisplayBeamProfile(BeamProfileVariables,Sensor);
        PrintInfo(Sensor);
      end;
    end;
  end;

function CoarseFindBeam(Sensor: Integer): Boolean;
var
  OldDisplay: Boolean;
begin
  OldDisplay := Display;
  Display := false;

SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],Ysp[S
ensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] +
0.5){BkgndThreshold},1{SpiralDelay},
    1{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind) then
begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
```

```
  Display := OldDisplay;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin

SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],Ysp[S
ensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] +
0.5){BkgndThreshold},100{SpiralDelay},
    2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,Verify2MilHole)
then begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  FineFindBeam := BeamFound;
end;

function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) and not AbortFlag then
    BeamFound := FineFindBeam(Sensor);
  FindLaserBeam := BeamFound;
end;

function FindHoles2(Sensor: Integer): Boolean;
{
  When we want to use new starting point
   updates hole position
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;
```

```
function FindHoles(Sensor: Integer): Boolean;
{
  Updates hole position
  Needs to confirm that we have 2 mil hole
}
var
  BkRead: LongInt;
  ADC,row,col: Integer;
begin
  FlushKey;
  Center('Searching for Sensor #'+IntStr(Sensor)+':
press any key to abort',25);
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
  BkRead := 0;
  for col:=1 to ColMax do begin
    for row:=1 to RowMax do begin
      ADC := ReadLaserBoard(S  sorChannel[Sensor]);
      inc(BkRead,ADC);
    end;
  end;
  SensorBkgnd[Sensor] := BkRead;
  { remove noise for centering }
  Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax) + 3;
  FindHoles := FindHoles2(Sensor);
end;
^Z
```

```
{-------------------------------------------------------
                    BEAM Utility Routines
-------------------------------------------------------}

procedure beep;
begin
  Sound(1000);
  delay(300);
  NoSound;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;
```

```
function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
  Str(num,temp);
  LongStr := temp;
end;

function RealStr(num: Real;AfterDecimal: Integer): String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal,temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
```

```
  spc := copy(blank,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;
```

```
{
  Find any char of Chars in Main starting at index Start
  Returns length(Main)+1 if no character of Chars in Main
}
function   FindStr(Start:   Integer;Main:   string;Chars:
str16): Integer;
var
  i,j,min: Integer;
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
  for i:=1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer): Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function WordVal(line: String;var code: Integer): Word;
var
  code2: Integer;
  num: Word;
begin
```

```
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;

function LongVal(line: String;var code: Integer):
  LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
var
  ofs,code2: Integer;
  num: Real;
begin
  ofs := 0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  RealVal := num;
end;

{
  Pick out a number from a string starting at Index and
  going to the next space or comma or end-of-string
}
```

```
procedure UnitValShortInt(var Storage: Integer;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure UnitValInt(var Storage: LongInt;Text:
  String;var Index: Integer);
var
  i: Integer;
begin
  Storage := LongVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure UnitVal(var Storage: Real;Text: String;var
Index: Integer);
var
  i: Integer;
begin
  Storage := RealVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

{
  Pick out text from a string starting at Index and going
  to the next comma or end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
  Integer);
var
```

```
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

{
  Show error and store it on disk
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'BEAM: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{ Harry's ultimate Draw-A-Box on the text screen routine
}
procedure DrawBox(x1,y1,x2,y2: integer;BoxType: integer);
type
  BoxMode = array[1..4] of Char;
```

```
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                        {  Type          1     2     3     4      }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then begin
      if   y1=y2   then   FullHorzBar   :=
Many(HorzBar[BoxType],x2-x1+1)
      else FullHorzBar := Many(HorzBar[BoxType],x2-x1-1);
    end else FullHorzBar := '';
  if y1<>y2 then begin
    if x1<>x2 then write(UpperLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(UpperRightCorner[BoxType]);
  end;
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
```

```
  if y1 = y2 then write(FullHorzBar)
  else begin
    if x1<>x2 then write(LowerLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(LowerRightCorner[BoxType]);
  end;
end;

{$F+}
procedure ExitBeam;
{$F-}
begin
  TextMode(C80);
  if (ExitCode > 0) and (ExitCode <> 255) then begin
    write('exit    code    ',ExitCode,'    at    addr
',Ofs(ErrorAddr));
    key := ReadKey;
  end;
  ExitProc := OldExitProc;
end;

procedure ClearLine(row: Integer);
begin
  gotoxy(1,row);
  write(blank);
end;

procedure Center(Text: String;Line: Integer);
begin
  gotoxy(1,Line);
  write(Blank);
  gotoxy((79-length(Text)) div 2,Line);
  write(Text);
end;

procedure PrintXY(x,y: Real);
var
```

```
  s: String;
begin
  if x < 0     then x := 0;
  if x > 65535 then x := 65535;
  if y < 0     then y := 0;
  if y > 65535 then y := 65535;
  s := '(' + WordStr(trunc(x)) + ',' + WordStr(trunc(y))
+ ')';
  if length(s)<13 then s := spc((13-length(s)) div 2) + s;
  write(s,spc(13-length(s)));
end;

procedure PrintPowerPos(Sensor: Integer);
begin
  ClearLine(23);
  gotoxy(10,23);
  write('Sensor #',Sensor,'Power',Power[Sensor]:5:1,' mW,
Center');
  PrintXY(Xsp[Sensor],Ysp[Sensor]);
end;

procedure PrintInfo(Sensor: Integer);
begin
  gotoxy(71,3);
  write('Power');
  gotoxy(69,4);
  write(Power[Sensor]:5:1,' mW');
  gotoxy(71,6);
  write('Center');
  gotoxy(68,7);
  PrintXY(Xsp[Sensor],Ysp[Sensor]);
  gotoxy(71,18);
  write('Maximum');
  gotoxy(73,19);
  write(BeamProfileVariables[Sensor].MaxSensorReading);
end;
```

```
procedure WaitKey;
begin
  Center('<Hit any key to continue>',25);
  repeat
  until keypressed;
  gotoxy(25,1);
  write(blank);
end;

procedure Pause(usec: LongInt);
var
  i,msec: LongInt;
begin
  i := 1;
  key := ' ';
  msec := trunc(round(usec / 1000));
  while (i < msec) and not keypressed do begin
    delay(1);
    inc(i);
  end;
  if keypressed then key := ReadKey;
end;

procedure FlushKey;
begin
  while keypressed do key := ReadKey;
end;

procedure ClearCol(col: Integer);
var
  col2: Integer;
begin
  if (col > 0) and (OpMode = GraphicsOp) then begin
    col2 := (col-1)*24-4;
    if col2<0 then col2 := 0;
    SetFillStyle(EmptyFill,0);
    Bar(col2,0,col2+23,330);
```

```
  end;
end;
^Z
```

```
{
  Beam Profiler monitoring routines
}

function ReadLaserBoard(Channel: Integer): Integer;
var
  count: LongInt;
  ADC,i: Integer;
begin
  count := Timeout;
  port[LaserAddr+12] := Channel;
  { must wait 10 us for EOS to go LO (286 @ 4us/loop) }
  i := i;
  while i < 10 do inc(i);
  repeat { then approx 110 us for conversion with signal
    D7 HI }
  dec(count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if PeakReading < ADC then PeakReading := ADC;
  if count = 0 then ReadLaserBoard := -1
    else ReadLaserBoard := ADC;
end;

procedure ReadTemperature(var Resist,Temper: Real);
var
  temp: Real;
  It,ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board use
  1/T, LOG(R) function TQ linearize the interpolation
}
```

```
  ADC := ReadLaserBoard(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC); { from 20K
  resistor on JB Board }
  if ADC > 214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] > Resist)
    do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It]));
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) -  ln(Resist)));
  end;
  Temper := Temper - 273.15;
end;

procedure println(text: String);
begin
  if not transparent then writeln(text);
end;

{-------------------------------------------------------
-                 BEAM SCANNING ROUTINES
-------------------------------------------------------
                        *************
                        * MOVE BEAM *
                        *************
  Routine to move mirrors to X,Y directly
  Note:  X,Y can be any kind of variable, we change to
    integers and get ADC% (for speed: later fixed addrs
    routines for each)
}
```

```
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 then pos := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
    { x is bit 0, y is bi
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInXAxis(x: LongInt);
begin
  AxisMove(x,1);
end;

procedure MoveInYAxis(y: LongInt);
begin
  AxisMove(y,2);
end;

procedure MoveBeam(x,y6: LongInt);
begin
  MoveInXAxis(x);
  MoveInYAxis(y);
end;

{
                    ******************
                    * GetBeamProfile *
                    ******************
}

procedure SetupBeamProfile(var BeamProfileVars:
  BeamProfileVarType;
  Sensor: Integer;X,t: Real;_ScanStep,_BkgndThreshold:
    Integer);
begin
```

```
  with BeamProfileVars[Sensor] do begin
    if X                >= 0 then Xpos           :=
trunc(X);
    if Y                >= 0 then Ypos           :=
trunc(Y);
    if _ScanStep        >= 0 then ScanStep       :=
_ScanStep;
    if _BkgndThreshold >= 0 then BkgndThreshold :=
_BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;
  procedure GetBeamProfile(var BeamProfileVars:
    BeamProfileVarType; Sensor: Integer);
  var
    Pass,row,col,ADC,delay,waste: Integer;
    Xcorner,Ycorner: LongInt;
  begin
    with BeamProfileVars[Sensor] do begin
      XYweight  :=0;
      Xweight   :=0;
      Yweight   :=0;
      TotalWeight  :=0; {for power, not removing raise
       till end}  :=0;
      SaturationCount  :=0;
      HiPoints  :=0;
      BeamFound := false;
      for col:=1 to ColMax do ColWeight[col] := 0;
      for row:=1 to RowMax do RowWeight[row] := 0;
      Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
      Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
      MoveBeam(Xcorner,Ycorner);
      waste := 0;
      while waste < Initial Delay do inc(waste);
      Peak Reading :=0;
```

```
        for Col:=1 to ColMax do begin {prefer to return
         around edges after line}
        X pos := Xcorner + col * ScanStep;
        X pos := Ycorner;
        MoveBeam(Xpos,Ypos);
        waste := 0;
        while waste < Col to Col Delay do inc(waste);
        for row:-1 to RowMax do begin
          inc(Ypos,ScanStep);
          MoveInYAxis(Ypos);
          waste := 0;
          while waste < RowToRowDelay do inc(waste);
          ADC := ReadLaserBoard(SensorChannel[Sensor]);
          if ADC < BkgndThreshold then ADC := 0
            else begin
              inc(HiPoints);
              BeamFound := true;
            end;
          if ADC > 210 then inc(SaturationCount);
          inc(TotalWeight,ADC);
          inc(ColWeight[col],ADC);
          inc(RowWeight[row],ADC);
          BeamMap[row,col] := ADC;
          inc(XYweight,ADC);
          inc(Xweight,Xpos * ADC); {want floating point
numbers here}
          inc(Yweight,Ypos * ADC);
        end; {for row}
      end; {for col}
      MaxSensorReading := PeakReading;
    end;
end;

{
                          ************
                          * GoodRead *
                          ************
```

```
}

procedure GoodRead(Sensor,iter: Integer);
var
  Try: Integer;
begin
  with BeamProfileVariables[Sensor] do begin
    Xaverage  := 0;
    Yaverage  := 0;
    TotWtAvg  := 0;
    GoodCount := 0;
    for Try := 1 to iter do begin
      SetupBeamProfile(BeamProfileVariables,Sensor,Xsp
        [Sensor],Ysp[Sensor],-1{ScanStep},
        -1{BkgndThreshold});
      GetBeamProfile(BeamProfileVariables,Sensor);
      if XYweight > 0 then begin
        Xaverage := Xweight / XYweight + Xaverage
        Yaverage := Yweight / XYweight + Yaverage
        TotWtAvg := TotWtAvg + TotalWeight -
          SensorBkgnd[Sensor];
        inc(GoodCount);
      end;
    end; {for Try}
    if GoodCount > 0 then begin
      Xaverage := Xaverage / GoodCount;
      Yaverage := Yaverage / GoodCount;
      TotWtAvg := TotWtAvg / GoodCount;
    end;
    Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
  end;
end;

{-------------------------------------------------------
                    HARDWARE ROUTINES
-------------------------------------------------------}
```

```
procedure IOSetup;
begin
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
end;

{
                        ************
                        * FindBeam *
                        ************
}

procedure SetupFindBeam(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer;X,Y: Real;_SpiralSideMax,_SpiralTries,
    _ScanStep,_BkgndThreshold,_SpiralDelay,_NumCenterPas
    ses: Integer);
begin
  with FindBeamVars[Sensor] do begin
  if X                 >= 0 then Xval             :=
trunc(X);
  if Y                 >= 0 then Yval             :=
trunc(Y);
  if _SpiralSideMax    >= 0 then SpiralSideMax    :=
_SpiralSideMax;
  if _SpiralTries      >= 0 then SpiralTries      :=
_SpiralTries;
  if _ScanStep         >= 0 then ScanStep         :=
_ScanStep
  if _SpiralDelay      >= 0 then SpiralDelay      :=
_SpiralDelay;
  if _BkgndThreshold   >= 0 then BkgndThreshold   :=
_BkgndThreshold;
  if _NumCenterPasses >= 0 then NumCenterPassses :=
_NumCenterPasses;
```

```
end;
end;

  function SpiralSearchForHole(var FindBeamVars:
    FindBeamVarType; Sensor: Integer): Boolean;
  var
    i,ADC,SpiralSide,waste: Integer;
    X,Y: LongInt;
  begin
    with FindBeamVars[Sensor] do begin
      X := trunc(Xval);
      Y := trunc(Yval);
      MoveBeam(X,Y);
      delay(SpiralDelay);
      BeamFound := false;
      SpiralSide := 1;
      while (SpiralSide <= SpiralSideMax) and not
        BeamFound do begin
         i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInXAxis(X);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
          else inc(X,ScanStep);
        inc(i);
      end;
         i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInYAxis(Y);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
          else inc(Y,ScanStep);
        inc(i);
```

```
      end;
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := Beam Found;
  end;
end;

  function FindBeam(var FindBeamVars:
    FindBeamVarType;Sensor: Integer; FindMethod:
    FindMethodType): Boolean;
  const
    HoleSeparation = 600;
  var
    i,ADC,pass,CantFind,SpiralSide: Integer;
    XspAv,YspAv: Real;
    X,Y,GoodX,GoodY: LongInt;
    BeamFound: Boolean;
  begin
    FindBeam := true;
    with FindBeamVars[Sensor] do begin
      CantFind := 0;
      repeat
        X := trunc(Xval);
        Y := trunc(Yval);
        MoveBeam(X,Y); { gets us to best starting
position }
{
  now spiral to search for holes
}
        SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
          {SpiralSideMax},-1{SpiralTries},-1{ScanStep},
```

```
            -1{BkgndThreshold},-1{SpiralDelay},-1
            {NumCenterPasses});
          BeamFound :=
            SpiralSearchForHole(FindBeamVariables,
            Sensor); if not BeamFound then begin
            SetupFindBeam(FindBeamVariables,Sensor,X,Y,
              SpiralSideMax * 2, -1{SpiralTries},-
              1{ScanStep}, BkgndThreshold -1,-
              1{SpiralDelay},-1{NumCenterPasses});
            inc(CantFind);
          end;
        until (CantFind >= SpiralTries) or BeamFound;
        if CantFind >= SpiralTries then begin
          MoveBeam(Xval,Yval);
          FindBeam := false;
          exit;
        end;
        X := Xval;
        Y := Yval;
      {
        Verify that we have 2 mil hole...

        ½ mil   1 mil ) left rear   2 mil   4 mil ) right
      front
                      ) of vat                    ) of vat
        4 mil   2 mil )             1 mil   ½ mil )

              Sensor #1                       Sensor #2

        We have the position of one of the four holes:  ½, 1,
        2, or 4 mils.  We want to find the 2 mil hole closest
        to the vat.  Note that the orientation of the holes is
        different for Sensors 1 and 2.

        Hole Size Check Procedure:
```

```
    1.  We may have the position for the ½ or 1 mil hole.
        Move towards the vat in the X-axis and search for
        the 1 or 2 mil hole.  If the beam is seen, adopt
        the new coordinates.

    2.  Now we may have the position for the 4 mil hole.
        Move towards the vat in the Y-axis and search for
        the 2 mil hole.  If the beam is seen, use the new
        coordinates.
}
  if (Sensor < 3) and (FindMethod = Verify2MilHole) then
  begin
    GoodX := X;
    GoodY := Y;
    if X < 32767 then inc(X,HoleSeparation)
      else dec(X,HoleSeparation);
    SetupFindBeam(FindBeamVariables,Sensor,X,Y,SpiralSid
      eMax div 2, -1{SpiralTries},-1{ScanStep},-
      1{BkgndThreshold},-1{SpiralDelay},-
      1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
        then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
    X := GoodX;
    Y := GoodY;
    if Y < 32767 then inc(Y,HoleSeparation)
      else dec(Y,HoleSeparation);
    SetupFindBeam(FindBeamVariables,Sensor,X,Y,-
      1{SpiralSideMax},-1{SpiralTries},-1{ScanStep},-
      1{BkgndThreshold},-1{SpiralDelay},-
      1{NumCenterPasses});
    if SpiralSearchForHole(FindBeamVariables,Sensor)
      then begin
      GoodX := Xval;
      GoodY := Yval;
```

```
      end;
      X := GoodX;
      Y := GoodY;
    end;

    XspAv := 0;
    YspAv := 0;
    if BeamFound then begin
      for pass:=1 to NumCenterPasses do with
        BeamProfileVariables[Sensor] do begin
        CantFind := 0
        { discard the first }
        SetupBeamProfile(BeamProfileVariables,Sensor,X,Y,4
          (ScanStep),trunc(Bk[Sensor] + 0.5));
        repeat
          GetBeamProfile(BeamProfileVariables,Sensor);
          if XYweight = 0 then inc(CantFind);
        until (XYweight > 0) or (Cantfind > 3);
        if CantFind > 3 then begin
          FindBeam := false;
          exit;
        end;
{
    Later select sensor hole size automatically
}
          X := trunc(Xweight / XYweight); { better
coordinates }
          Y := trunc(Yweight / XYweight);
          Power[Sensor] := (TotalWeight -
SensorBkgnd[Sensor])
            * (SensorCalib[Sensor,3]);
          XspAv := XspAv + Xweight / XYweight;
          YspAv := YspAv + Yweight / XYweight;
        end; {for pass}
        if NumCenterPasses > 1 then begin
          XspAv := XspAv / NumCenterPasses;
          YspAv := YspAv / NumCenterPasses;
```

```
      end;
    end; {if BeamFound}
    Xval := X;      { update integer coordinates }
    Yval := Y;
    Xavg := XspAv;  { update average coordinates }
    Yavg := YspAv;
  end;
end;

function CoarseFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor],50{SpiralSideMax},3{SpiralTries},16{Scan
    Step}, trunc(Bk[Sensor] +
    0.5){BkgndThreshold},1{SpiralDelay},
    1{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind) then
    begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor],50{SpiralSideMax},3{SpiralTries},16{Scan
    Step},trunc(Bk[Sensor] +
    0.5){BkgndThreshold},100{SpiralDelay},
    3{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,Verify2MilHole)
    then begin Xsp[Sensor] :=
    FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  FineFindBeam := BeamFound;
```

```
end;

function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) then
    BeamFound := FineFindBeam(Sensor)
  FindLaserBeam := BeamFound;
end;

{ - - - - - }

function FindHoles2(Sensor: Integer): Boolean;
{
  When we want to use new starting point
    update hole position
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;

function FindHoles(Sensor: Integer): Boolean;
{
  Updates hole position
  Needs to confirm that we have 2 mil hole
}
var
  BkRead: LongInt;
  ADC,row,col: Integer;
begin
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
  BkRead := 0;
  for col := 1 to ColMax do begin
    for row := 1 to RowMax do begin
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
      inc(BkRead,ADC);
```

```
    end;
  end;
  SensorBkgnd[Sensor] := BkRead;
  { remove noise for centering }
  Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax) + 3;
  FindHoles := FindHoles2(Sensor);
end;

{=====================================================}

{
  Read BEAM.PRM parameters file
}
procedure ReadBeamParamsFile;
var
  ParamFile: text;
  1,count,code,code1,code2,Sensor,Pinhole: Integer;
    cmd,ParamLine: String;

  procedure GetParamFileLine;
  begin
    if eof(ParamFile) then ParamLine := ''
    else repeat
      readin(ParamFile,ParamLine);
    until eof(ParamFile) or ((ParamLine <> '') and
      (ParamLine[1] <> '!'));
  end;

begin
  {$I-}
  assign(ParamFile,BeamParamFileName);
  reset(ParamFile);
  if IOresult<>0 then begin
    assign(ParamFile,HomeDirectory+BeamParamFileName);
    reset(ParamFile);
    if IOresult<>0 then halt(10);
  end;
```

```
{$I+}
GetParamFileLine;
if copy(ParamLine,1,1) = chr(34) then begin { handle
  quoted options line }
  i := pos(chr(34),copy(ParamLine,2,255));
  if i>0 then ParamLine[0] := chr(i);
  ParamLine := NoQuotes(ParamLine);
  GetParamFileLine;
end;
XRest := WordVal(ParamLine,code);
YRest := WordVal(copy(ParamLine,code+1,255),code;
GetParamFileLine;
SensorNum := WordVal(ParamLine,code);
for Sensor := 1 to SensorNum do begin
  GetParamFileLine;
  SensorChannel[Sensor] := WordVal(ParamLine,code);
  GetParamFileLine;
  XSensorPos[Sensor] := WordVal(ParamLine,code);
  YSensorPos[Sensor] :=
    WordVal(copy(ParamLine,code+1,255), code);
  for Pinhole:=1 to 4 do begin
    GetParamFileLine;
    SensorCalib[Sensor,Pinhole] :=
RealVal(ParamLine,code);
  end;
  end; {for Sensor}
  GetParamFileLine;
  TempChannel := WordVal(ParamLine,code); { assume
    only one thermometer at fir
  GetParamFileLine;
  TempCalPts := WordVal (ParamLine,code);
  for i:=1 to TempCalPts do begin
    { Resistance[i],Temperature[i] may not be integers
}
    GetParamFileLine;
    Resistance [i] := RealVal(ParamLine,code);
```

```
      Temperature[i] :=
        RealVal(copy(ParamLine,code+1,255), code) +
        273.15;
    end;
    for i:=1 to 3 do begin { read in Fast, Medium, &
      Slow Calib Delay values }
    GetParamFileLine;
    CalibDelays[i,1] := LongVal(ParamLine,code1);
    CalibDelays[i,2] :=
      trunc(RealVal(copy(ParamLine,code1+1,
      255),code2)*10+0.5)
    CalibDelays[i,3] :=
      trunc(RealVal(copy(ParamLine,code2+ code1
      +1,255),code)*10 + 0.5)
  end;
  close(ParamFile);
  Resistance  [0] := Resistance [1] * 2 - Resistance
[2];
  Temperature [0] := Temperature[1] * 2 -
Temperature[2];
end;

{-------------------------------------------------------}

procedure SetupMonitoring;
begin
  port[LaserAddr+3] := $88;
  for i:=1 to 2 do begin
    Xsp[i] := XSensorPos[i];
    Ysp[i] := YSensorPos[i];
  end;
end;

procedure CantLocateSensor(num: Integer);
begin
  PrintError('Couldn''t locate Profiler Sensor #' +
IntStr(num) + ' on layer ' + IntStr(ControlLayer));
```

```
end;

procedure MonitorLaserBeamDrift;
type
  BufType = array[1..19] of Byte;
const
  CmdBufMask: BufType = (STEREO_D1,0,0,0,0,0,0,0,0,
                         STEREO_D2,0,0,0,0,0,0,0,0,0);
var
  CmdBuf: BufType;
  i: Integer;
begin
  if not (NoDriftFlag or LaserOff) then begin
    Action('ANALYZING');
    CmdBuf := CmdBufMask;
    if FindHoles(1) then begin
      with FindBeamVariables[1] do begin
        Xsp[1] := Xavg;    { fast locate next time }
        Ysp[1] := Yavg;
        ConvertRtoS(@Xavg,@CmdBuf[2]);
        ConvertRtoS(@Yavg,@CmdBuf[6]);
      end;
      if FindHoles(2) then begin
        with FindBeamVariables[2] do begin
          Xsp[2] := Xavg;
          Ysp[2] := Yavg;
          ConvertRtoS(@Xavg,@CmdBuf[11]);
          ConvertRtoS(@Yavg,@CmdBuf[15]);
        end;
        ShowError(BinarySTEREO(@CmdBuf));
      end else CantLocateSensor(2);
    end else CantLocateSensor(1);
  end;
end;
```

```
BUILD.PAS
{$R-,S+,I+,D+,T-,F-V-,B-,N-,L+}
{$M 16384,0,655360}

{============================================================
  3D Systems Stereolithography System Software
============================================================

BUILD.PAS
SLA-1 Stereolithography Part-Builder
------------------------------------------------------------

BUILD-specific Commands:

  ZW secs     - set Z-axis Wait time in seconds
  ZD distance - set Z-axis dip Depth in millimeters
  ZV velocity - set Z-axis Velocity parameter
  ZA accel    - set Z-axis Acceleration parameter
------------------------------------------------------------

Recent History:
  9/ 1/87   Ver 2.04  Version 2 released
  9/ 2/87   Ver 2.06  variable dip logic corrected
  9/ 3/87   Ver 2.07  compiled by TBM (Modified Turbo
                      Basic) double parse of options
                      EX & CL before double EC to
                      suppress glitch
  9/ 4/87   Ver 2.08  added SV, SI critical areas
  9/17/87   Ver 2.10  PRM numbers less than 0.1
                      supported selectable range of Z
                      layers with /START, /STOP options
                      keyboard pause / continue / skip
                      to layer control
  9/22/87   Ver 2.11  fixed incorrect day of week read
  9/23/87   Ver 2.12  new SUPER commands ZW, ZD, ZV and
                      ZA structural changes to SUPER.PRM
```

9/25/87	Ver 2.20	new SLICE 19 block format supported
9/28/87	Ver 2.21	/NEGZ Z-stage is negative flag default option line in SUPER.PRM added NFUB support added lower-case SUPER command support
9/30/87	Ver 2.25	no execute for empty blocks (primarily for L block)
10/20/87	Ver 2.26	/DEMO out-of-vat-delay option for AutoFact '87
10/26/87	Ver 2.27	*.P files now *.L file error file support
11/03/87	Ver 2.28	special demo screen read from file SUPER.DMO
11/04/87	Ver 2.29	demo action descriptions with good timing
11/12/87	Ver 2.30	first Beta software release
11/15/87	Ver 2.31	added directory of *.L files
11/20/87	Ver 2.32	multiple part-making capability added
12/04/87	Ver 2.33	slice20 FUB FDB blocks Menu System forced pause at end of part-making run BLOCK.ID file supported
12/16/87	Ver 2.40	SUPER converted from Turbo Basic to Turbo Pascal faster file transveral, laser buffer filling no special demo screen support
12/18/87	Ver .41	fixed Z-Table dip routine (inf board output)
12/21/87	Ver 2.50	graphics support added start-up and part-making information displays laser vectors shown on screen

12/23/87	Ver 2.51	Block File read speed up skipping to new layers sped up 150 times
12/28/87	Ver 2.52	skip to new layer fixed Layer Control File command processing fixed added ability to flip between the two graphic screens
1/06/88	Ver 2.53	break abort allowed during skip action negative offsets shown as such wait action F1/F2 support added graphics display of error messages axes orientation arrows for view window better file directory display
1/22/88	Ver 2.54	control of min/max vat viewport coords via SUPER.PRM ShowFiles based on *.V instead of *.L case no longer important on SUPER.PRM option line better non-existent input files recovery
1/25/88	Ver 2.60	Graphics Supervisor no longer classified as Experimental support of *.R Range Control Files (as well as *.L) correct layer count derived from Range Control File
2/06/88	-	/DEMO option made functional rivet processing made functional
3/08/88	Ver 2.80	added copyright notice installed gain and offset drift correction implemented binary STEREO support
4/04/88	Ver 3.00	change messages and menus

```
  4/08/88   Ver 3.01  added binary Vector File support
                      removed Critical Area feature
                      SUPER renamed to BUILD
  4/11/88   Ver 3.02  fixed EofVectorFile routine
  4/13/88   Ver 3.03  fixed skipped-to layer block
                      command default problem
  4/15/88   Ver 3.04  added layer number to profiling
                      error messages
                      default part name input from last
                      part PREPARED still remaining to
                      do...

                      part start and completion time
                      logged to history file support of
                      part-specific build parameters
                      warning message if laser not on or
                      shutters closed set up of laser-
                      off control through Menu program
                      (not for /LOFF)

-------------------------------------------------------}

uses Dos,
     Crt,
     Graph,    { library of graphics routines }
     Drivers,  { all the BGI drivers }
     Fonts;    { all the BGI fonts }

type
  str4  = string[4];
  str6  = string[6];
  str10 = string[10];
  str16 = string[16];
  str40 = string[40];
  str80 = string[80];
OpModeType = (textOp,GraphicsOp);
FpPointerStruct = array[0..255] of Byte;
```

```
FpPointerType    = ^FpPointerStruct;
BeamProfileVarSensorType = record
  Xpos             : LongInt;  {  inputs  }
  Ypos             : LongInt;
  ScanStep         : Integer;
  BkgndThreshold   : Integer;
  InitialDelay     : Integer;
  RowToRowDelay    : Integer;
  ColToColDelay    : Integer;
  XYweight         : LongInt;  { outputs }
  Xweight          : LongInt;
  Yweight          : LongInt;
  TotalWeight      : LongInt;
  HiPoints         : Word;
  SaturationCount  : Integer;
  MaxSensorReading: Integer;
  BeamMap          : array[0..30,0..30] of Integer;
  RowWeight        : array[0..30] of Word;
  ColWeight        : array[0..30] of Word;
end;
BeamProfileVarType = array [1..3] of
BeamProfileVarSensorType;
FindBeamVarSensorType = record
  Xval             : LongInt;  { inputs & updated }
  Yval             : LongInt;
  SpiralDelay      : Word;     { inputs }
  SpiralSideMax    : Integer;
  SpiralTries      : Integer;
  ScanStep         : Integer;
  BkgndThreshold   : Integer;
  NumCenterPasses : Integer;
  Xavg             : Real;     { outputs }
  Yavg             : Real;
end;
FindBeamVarType = array[1..3] of FindBeamVarSensorType;
FindMethodType = (none,NormalFind,Verify2MilHole);
```

```
var
  FirstVar: Byte; { must be first variable defined }
  Ch,key: Char;
  index,dummy,code,i,j,k: Integer;
  WeekDay,Hour,Minu,Sec,Mon,Day,Year: Integer;
  temp,cmd,SingleBar: str80;
  ShowStuff,ZTableOff,DipFlag,NegZStage,Demo: Boolean;
  FirstLayer,SameBlock,MoveOnFlag,SkipFlag,PauseFlag:
Boolean;
  VectorFileIsBinary,NoDriftFlag: Boolean
  QSize,Layer,VPosition,StartZ,EndZ,NumLayers,LayersTo
 Build:  Long nt;
  PauseAfterLayer: LongInt;
  XYScale,ZPitch,PartHeight: Real;
  CurrentDate,CurrentTime: str16;

ParamOptions,ControlLine,VectorLine,PartName,DefaultPart
    Name: str80;
  StartTimeStamp,EndTimeStamp: str40;
  BlockMnemonicsCount,OutOfVatDelay,x1,y1,x2,y2: Word;

VectorFileLoc,MaxLaserQueueBlockSize,LaserQueueBlockSize
:
    LongInt;
  Binary_X1,Binary_Y1,Binary_X2,Binary_Y2: Word;
  regs:  Registers;
  LastPartFile,ControlFile,ErrorFile: Text;
  VectorFile: File;  {untyped}
  XOffset,YOffset,ZAddress: LongInt;
  ParVectorType: array[0..100] of str10;
  ParStereoDef: array[0..100] of str80;
  BlockMnemonics: array[1..20] of str10;
  Offsets: array[0..50,1..2] of LongInt;
  LayerCount,NumXOffsets,NumYOffsets,NumOffsets: Word;
  ParCount,RelaxationTime: Word;
  LaserVersion,StereoVersion: str4;
  check1,check2: Boolean;
```

```
  VectorNumRead,BufIndex,ControlIndex,BlockIndex:
Integer;
  SaveSec,NewSec,DeltaSecs,OldDeltaSecs,ControlLayer,
    VectorLayer: LongInt;
  GrDeltaX,GrDeltaY: LongInt;
  ControlType,VectorType: str10;
  DipDepth,ZTableVeloc,ZTableAccel: Real;
  SliceRes,ZSpacing,Thickness: Real;
  OnScreen,GraphDriver,GraphMode,Error: Integer;
  ViewPort: ViewPortType;
  OpMode: OpModeType;
  LayerCtrl,SwapXY,NegX,NegY: Boolean;
  GrMinX,GrMinY,GrMaxX,GrMaxY:  LongInt;
  RangeCount,BlockCount: Integer;
  BlockVectorCount,RangeZStart,RangeZEnd: LongInt;
  RangeZSpace,RangeCureDepth: Integer;
  ZStart,ZEnd: array[1..100] of LongInt;
    { all the ranges }
  ZSpace,CureDepth: array [1..100] of Integer; {
   }
  VectorTypes: array[1..100] of str6;
    { just for one
    range }
  BlockCmds: array[1..100] of str80;        {       }

  { - PROFILING VARIABLES - }
  { beam arrays}
  Resistance,Temperature: array[0..100] of Real;
  { sensor arrays }
  SensorChannel,XSensorPos,YSensorPos,SensorBkgnd:
array[1..3] of Word;
  Power,Bk: array[1..3] of Real;
  Xsp,Ysp: array[1..3] of Real;
  SensorCalib: array[1..3,1..4] of Real;
  LastField: array[1..4] of Integer;
  { miscellaneous }
  Transparent,BeamFound: Boolean;
```

```
    XRest,YRest,TempChannel,TempCalPts: Word;
    GoodCount,PeakReading,Sensor,Select,SensorNum:
Integer;
    Xpos,Ypos: LongInt;
    Xaverage,Yaverage,TotWtAvg: Real;
    OldExitProc: Pointer;
    BeamProfileVariables: BeamProfileVarType;
    FindBeamVariables: FindBeamVarType;
    CalibDelays: array[1..3,1..3] of LongInt;
    LastVar: Byte;  { must be last varialbe-to-be-zeroed
      defined }

    VectorBuf: array[1..4100] of Char;

const
    VersionId      = '3.04';
    MSDOS          = $21;
    LASERVECTOR    = $64;
    STEREOVECTOR   = $65;
    BlockIdentifiers  = 'ZXYISHVAE CKEFGK';
    ErrorFileName     = 'C:\SYSTEM.ERR';
    SystemFileName    = 'C:\SLA1.SYS';
    HomeDirectory     = 'C:\3DSYS\';
    BlockIdFileName   = 'BLOCK.ID';
    ParamFileName     = 'BUILD.PRM';
    BeamParamFileName = 'BEAM.PRM';
    LastpartFileName  = 'PART.LST';
    BuildCommandCount = 4;
    BuildCommands: array[1..4] of str4=('ZW', 'ZD',
      'ZV', 'ZA');
    Ctrl C = ^C;
    Bs     = ^H;
    Tab    = ^I;
    Lf     = ^J;
    Cr     = ^M;
    CtrlS = ^S;
    F1     = 59;
```

```
  F2    = 60;
  sp80  = '                                                    ';
  DayOfWeek: String[21] = 'SunMonTueWedThuFriSat';
  Zel: array-1..12] of Integer =
    (1,4,3,6,1,4,6,2,5,0,3,5);
  VectorRecordSize = 4096;
  PowerHistoryFileName    = 'C:\HISTORY\POWER.MON';
  ProfileHistoryFileName  = 'C:\HISTORY\PROFILE.MON';
  Blank = '
  ColMax  = 22;
  RowMax  = 22;
  LaserAddr   = $300
  NumSensors  = 2;
  Timeout     = 60000;

{-----------------------------------------------------}

procedure OverText(Text: String;len: Integer); forward;
procedure PrintError(Text: String);            forward;

{ include various files }

  {$i utility.inc}    { miscellaneous routines        }
  {$i keybrd.inc}     { keyboard routines             }
  {$i graph.inc}      { screen graphics routines      }
  {$i time.inc}       { date and time routines        }
  {$i zstage.inc}     { Z-stage elevator routines     }
  {$i lctrl.inc.}     { laser control routines        }
  {$i stercmds.inc}   { STEREO binary command equates }
  {$i bldmon.inc}     { profile monitor routines      }

{-----------------------------------------------------}

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
```

```
begin
  beep;
  if OpMode = TextOp then begin
    if whereX>1 then writeln;
    writeln('***',Text);
  end else begin
    MoveTo(3*8,24*14-2);
    OverText('***' + Text,55);
  end;
end;

function EofVectorFile: Boolean;
begin
  EofVectorFile := eof(VectorFile) and (BufIndex >
VectorNumRead);
end;

procedure OpenFiles;
var
  FirstChar: Char;
label EasyLayer;
begin
  { look for either a .R or .L control file }
  if LayerCtrl then goto EasyLayer;
  assign(ControlFile,PartName+'.R');
  {$I-}
  reset(ControlFile);
  if IOresult = 0 then begin
    if eof(ControlFile) then begin
      SystemError('Error:Control File'+ PartName+'.R
       empty.');
      halt(6);
    end;
  end else begin
EasyLayer:
  assign(ControlFile, PartName + '.L');
  reset(ControlFile);
```

```
  if IOresult<>0 then begin
    SystemError('No Control File for part '+ PartName);
    halt(5);
  end else begin
    if eof(ControlFile) then begin
      SystemError('Error:Control File ' + PartName +  '
      .L empty.');
      halt(7);
    end else LayerCtrl := true;
  end;
end;
  { look for the vector file }
assign(VectorFile,PartName + '.V');
reset(VectorFile,1);
if IOresult<>0 then begin
  CantOpen(PartName + '.V');
  halt(8);
end;
if EofVectorFile then begin
  SystemError('Error:' + PartName + '.V empty.');
  halt(9);
end;
{$I+}
BlockRead(VectorFile,FirstChar,1,VectorNumRead);
if FirstChar = #0 then VectorFileIsBinary := true
else begin
  VectorFileIsBinary := false;
  close(VectorFile);
  reset(VectorFile,1);
 end;
end;

procedure ParseControlLineComments;
begin
  if (length(ControlLine)>0) and
(copy(ControlLine,1,6)='
  !MERGE') then begin
```

```
    i := pos('-STARTLAYER ',ControlLine);
    if (i>0) and (StartZ=0)
      then StartZ :=
LongVal(copy(ControlLine,i+12,5),dummy);
    i := pos('-ENDLAYER ',ControlLine);
    if (i>0) and (End=65535)
      then EndZ := LongVal(copy(ControlLine,i+10,5),
       dummy);
    i := pos('-NUMLAYERS ',ControlLine);
    if (i>0) and (NumLayers=65535)
      then NumLayers := LongVal(copy(ControlLine,
       i+11,5),
      dummy);
    end;
end;

procedure GetLayerControlLine;
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      SameBlock := false;
      exit;
  end;
  readln(ControlFile,ControlLine);
  ParseControlLineComments;
until (length(ControlLine)>0) and (ControlLine[1]<>'!');
ControlIndex := 1;
UnitValInt(ControlLayer,ControlLine,ControlIndex);
UnitStr(ControlType,ControlLine,ControlIndex);
SameBlock := (VectorLayer = ControlLayer) and
  (VectorType =
 ControlType);
{ writeln(SameBlock,' CLayer=',ControlLayer,' CType=',
  ControlType,' CLine=', ControlLine}
end;
procedure GetRangeControlLine;
```

```
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      exit;
    end;
    readln(ControlFile, ControlLine);
    ParseControlLineComments;
  until (length(ControlLine)>0) and
(ControlLine[1]<>'!');
end;

procedure RedRange;
var
  Index,p: Integer;
  AtEnd: Boolean;
begin
  Index := 1;
  UnitValInt      (RangeZStart,     ControlLine,Index);
  UnitValInt      (RangeZEnd,       ControlLine,Index);
  UnitValShortInt (RangeZSpace,     ControlLine,Index);
  UnitValShortInt (RangeCureDepth,  ControlLine,Index);
AtEnd :=false;
BlockCount := 0;
repeat
  GetRangeControlLine;
  if (ControlLine='') or (ControlLine[1] in ['0'..'9'])
then
   AtEnd := true
    else begin
    inc(BlockCount);
    p := pos(',',ControlLine);
    if p=0 the p := length(ControlLine)+1;
    VectorTypes[BlockCount] := copy(ControlLine,1,p-1);
    BlockCmds [BlockCount] := NoQuotes(copy
    (ControlLine,p+1,255));
```

```
  end
  until AtEnd;
end;

procedure ReSeekVectorFile;
begin
  if VectorFileLoc  > FileSize(VectorFile)
    then VectorFileLoc := FileSize(VectorFile); { force
     eof condition}
    Seek(VectorFile,VectorFileLoc);
    BlockRead(VectorFile,VectorBuf,VectorRecordSize,
     VectorNumRead);
    BufIndex := 1;
    BlockVectorCount :=0;
function GetVectorCh: Char;
begin
  if not EofVector File then begin
      GetVectorCh :=,VectorBuf[BufIndex];
      inc(BufIndex);
      inc(VectorFileLoc);
      if BufIndex := 1;
      BlockRead(VectorFile,VectorBuf,VectorRecordSize,
       VectorNumRead);
    end;
  end;
end;

function GetVectorByte: Byte;
begin
  if not EofVectorFile then GetVectorByte :=
ord(GetVectorCh)
  else GetVectorByte :=0;
end;

function GetVectorWord: Word;
var
  WordValue: Word;
```

```
var
  WordValue: Word;
begin
  WordValue := GetVectorByte;
  GetVectorWord := WordValue + GetVectorByte shl 8;
end;

function GetVectorLongInt: LongInt;
var
  LongIntValue: LongInt;
begin
  LongIntValue := GetVectorWord;
  GetVectorLongInt := LongIntValue + GetVectorWord shl
16;
end;

procedure GetVectorEntryAscii;
var
  Ch: Char;
  store: Integer;
begin
  repeat
    VectorLine := '';
    if EofVectorFile then begin
      SameBlock := false;
      exit;
    end;
    Ch := GetVectorCh;
    VectorLine[1] := Ch;
    store :=2;
    while (not EofVectorFile) and (Ch<>Cr) do begin
      Ch := GetVectorCh;
      VectorLine[store] := Ch;
      inc(store);
    end;
    VectorLine[0] := chr(store);
    { throw away line-feed }
```

```
    if not EofVectorFile then Ch :=GetVectorCh;
    if VectorLine[1] in ['A'..'Z'] then begin
    BlockIndex := 1;
    UnitStr(VectorType,VectorLine,BlockIndex);
    UnitValInt(VectorLayer,VectorLine,BlockIndex);
  end;
until (length(VectorLine)>0) and (VectorLine[1]<>'!');
SameBlock := (VectorLayer = ControlLayer) and
  (VectorType =
 ControlType);
{ writeln(SameBlock,' BLayer=',VectorLayer,' BType=',
   VectorType,' BLine=',Vectc }
end;

procedure GetVectorEntryBinary;
begin
  if BlockVectorCount > 0 then begin
    Binary _X1 := GetVectorWord;
    Binary _Y1 := GetVectorWord;
    Binary _X2 := GetVectorWord;
    Binary _Y2 := GetVectorWord;
      dec(BlockVectorCount);
  end else begin
      VectorType := BlockMnemonics[GetVectorByte] +
       chr(ord('0') + GetVectorByte);
      VectorLayer := GetVectorWord;
      BlockVectorCount := GetVectorLongInt;
  end;
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType =
   ControlType);
end;
procedure GetVectorEntry;
begin
  if VectorFileIsBinary then GetVectorEntryBinary
    else GetVectorEntryAscii;
end;
```

```
{-------------------------------------------------------}

procedure BlockSkipAscii;
var
  Ch:  Char;
  i,NumRead: Integer;
  Mmatch: str10;
  SkipBuf: String;
begin
  Match := Lf+'L';
  while (VectorLayer < StartZ) and not EofVectorFile do
begin
    Seek(VectorFile,VectorFileLoc);
    BlockRead(VectorFile,SkipBuf[1],255,NumRead);
    i := pos(Match,SkipBuf);
    SkipBuf[0] := chr(255);
    while (i=0) and not(eof(VectorFile) or keypressed)
do
     begin
      inc(VectorFileLoc,NumRead);
      move(SkipBuf[236],SkipBuf[1],20);
      Blockread(VectorFile,SkipBuf[20],235,NumRead);
      if keypressed then HandleKeyboardRequest;
      i := pos(Match,SkipBuf);
      if i>0 then dec(i,19);
  end;
      if i>0 then inc(VectorFileLoc,i)
      else VectorFileLoc := FileSize(VectorFile);
      ReSeekVectorFile;
      GetVectorEntry;
      if keypressed then HandleKeyboardRequest;
    end;
end;

procedure BlockSkipBinary;
type
  VectorBlockEntryType = record
```

```
    Block       : Byte;
    MergeSet    : Byte;
    Layer       : Word;
    VectorCount : LongInt;
  end;
var
  NumRead: Integer;
  VectorBlockEntry: VectorBlockEntryType;
begin
  Seek(VectorFile,VectorFileLoc);
  while (VectorLayer < StartZ) and not eof(VectorFile)
do
   begin

BlockRead(VectorFile,VectorBlockEntry,SizeOf(VectorBlock
     Entry),NumRead);
    if VectorBlockEntry.Layer < StartZ then begin
      inc(VectorFileLoc,8 + VectorBlockEntry.
  VectorCount * 8);
      Seek(VectorFile,VectorFileLoc);
    end else VectorLayer := VectorBlockEntry.Layer;
    if keypressed then HandleKeyboardRequest;
  end;
  ReSeekVectorFile;
  GetVectorEntry;
end;

procedure BlockSkip;
begin
  if VectorFileIsBinary then BlockSkipBinary
    else BlockSkipAscii;
end;

{-----------------------------------------------------}

  Send execute command to the Stereo driver
}
```

```
procedure LaserExecute;
const
  BinaryCommand: array[1..5] of Byte =
    (STEREO_EX,STEREO_CL,STEREO_EC,STEREO_EC,0);
begin
  ShowError(BinarySTEREO(@BinaryCommand));
  LaserQueueBlockSize := 0;
end;

{-----------------------------------------------------}

Send laser control command(s) to the Stereo driver

}
procedure SendLaserCommand(cmd: String);
begin
  ShowError(STEREO(cmd));
LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
   LaserExecute;
end;

procedure SendBinaryLaserCommand(BufPtr: Pointer);
begin
  ShowError(BinarySTEREO(BufPtr));
  LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
   LaserExecute;
end;

-----------------------------------------------------

Pick Out and Execute BUILD-specific Command
}
procedure PickOutBuildCommand(var cmd: String);
var
  op: str4;
```

```
  k,index: Integer;
  arg: Real;
begin
  repeat
    op :=  upper(copy(cmd,1,2));
    index := 1;
    while (index<=BuildCommandCount) and
(pos(BuildCommands [index],op)=0)
      do inc(index);
    if index<=BuildCommandsCount then begin
      k := 3;
      while (k<length(cmd)) and (cmd[k]<>';') do inc(k);
      inc(k);
      arg := RealVal(copy(cmd,3,k-3),code);
      case index of
        1: {ZWait}  RelaxationTime  :=  trunc(arg);
        2: {ZDepth} DipDepth        :=  arg;
        3: {ZVeloc} ZTableVeloc     :=  arg;
        4: {ZAccel} ZTableAccel     :=  arg;
      end;
      cmd := copy(cmd,k,199);
    end;
  until index > BuildCommandsCount;
end;

-------------------------------------------------------

  Process both BUILD and STEREO commands

  BUILD and STEREO commands are broken apart and
  executed in the proper sequence, but by the two
  different programs.
}
procedure ProcessCommands(cmd: String);
var
  OrgCmd: String;
  i,j,BuildIndex: Integer;
```

```
begin
  cmd := upper(cmd);
  repeat
    OrgCmd := cmd;
    i := length(OrgCmd) + 1;
    for j:=1 to BuildCommandsCount do begin
      BuildIndex := pos(BuildCommands[j],OrgCmd);
      if (BuildIndex > 0) and (BuildIndex < i) then i :=

       BuildIndex;
    end;
    cmd := copy(OrgCmd,1,i-1);
    if length(cmd)>0 then SendLaserCommand(cmd);
    cmd := copy(OrgCmd,i,255);
    if length(cmd)>0 then PickOutBuildCommand(cmd);
  until length(cmd)=0;
end;

}-----------------------------------------------------}

{
  Find Block Type in ParVectorType array.  If there,
send the corresponding ParStereoDef default string to
Stereo.
}
procedure SetBlockDefaults;
begin
  i := 1;
  while (ParVectorType[i]<>ControlType) and
   (i<=ParCount) do
   inc(i);
  if i<=ParCount then
    ProcessCommands(ParStereoDef[i]);
end;

{-----------------------------------------------------}
```

```
procedure GraphVector;
var
  grx1,grx2,gry1,gry2,temp: Word;
begin
  if NegX then begin
    grx1 := round((65535-x1-XOffset-GrMinX)/GrDeltaX*
      (77-45)*8);
    grx2 := round((65535-x2-XOffset-GrMinX)/GrDeltaX*
      (77-45)*8);
  end else begin
    grx1 := round((x1+YOffset-GrMinX)/GrDeltaX*(77-
     45)*8);
    grx2 := round((x2+YOffset-GrMinX)/GrDeltaX*(77-
     45)*8);
  end;
  if NegY then begin
    gry1 := round((65535-y1-XOffset-GrMinY)/GrDeltaY*
      (22-9)*14);
    gry2 := round((65535-y2-XOffset-GrMinY)/GrDeltaY*
      (22-9)*14);
  end else begin
    gry1:=round((y1+YOffset-GrMinY)/GrDeltaY*(22-9)*14);
    gry2:=round((y2+YOffset-GrMinY)/GrDeltaY*(22-9)*14);
  end;
  if SwapXY then begin
    temp := grx1;
    grx1 := gry1;
    gry1 := temp;
    temp := grx2;
    grx2 := gry2;
    gry2 := temp;
  end;
  SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
  Line(grx1,gry1,grx2,gry2);
  SetViewPort(0,0,GetMaxX,GetMaxY,true);
end;
```

```
procedure SendVector(DrawMethod: Word);
type
  BinaryCommandType = array[1..11] of Byte;
const
  BinaryCommandMask: BinaryCommandType =
  (STEREO_JP,0,0,0,0,1,0,0,0,0,0,0);
var
  wd: Word;
  BinaryCommand: BinaryCommandType;
begin
  BinaryCommand := BinaryCommandMask;
  BinaryCommand[6] := DrawMethod;
  wd := trunc(x1 + XOffset);
  BinaryCommand[2] := lo(wd);
  BinaryCommand[3] := hi(wd);
  wd := trunc(y1 + YOffset);
  BinaryCommand[4]  := lo(wd);
  BinaryCommand[5]  := hi(wd);
  wd := trunc(x2 +  XOffset);
  BinaryCommand[7]  := lo(wd);
  BinaryCommand[8]  := hi(wd);
  wd := trunc(y2 +  YOffset);
  BinaryCommand[9]  := lo(wd);
  BinaryCommand[10] := hi(wd);
  SendBiaryLaserCommand(@BinaryCommand);
  if OpMode=GraphicsOp then GraphVector;
end;

procedure DrawVector;
begin
  SendVector(STEREO_MV);
end;

{-----------------------------------------------------}

procedure DrawSectionRivet;
begin
```

```
  if true {always} then DrawVector
  else SendVector(STEREO_RV);
  if OpMode=GraphicsOp then GraphVector;
end;

{-------------------------------------------------------}

procedure DrawSkinRivet;
begin
  if true {always} then DrawVector
  else SendVector(STEREO_RV);
  if OpMode=GraphicsOp then GraphVector;
end;
{=======================================================

  BLOCK PROCESSING ROUTINES:            SLICE

  ZBlock:  Section Border                 V1
  XBlock:  Section X Hatching             V1
  YBlock:  Section Y Hatching             V1
  IBlock:  Section Angle Hacthing         V18   8/27/87
  SBlock:  Up Facing Skin Border          V17
  HBlock:  Up Facing Skin X Fill          V17
  VBlock:  Up Facing Skin Y Fill          V17
  ABlock:  Up Facing Skin X Hatch         V17
  BBlock:  Up Facing Skin Y Hatch         V17
  JBlock:  Up Facing Skin Angle Hatch     V18
  CBlock:  Down Facing Skin Border        V17
  DBlock:  Down Facing Skin X Fill        V17
  EBlock:  Down Facing Skin Y Fill        V17
  FBlock:  Down Facing Skin X Hatch       V17
  GBlock:  Down Facing Skin Y Hatch       V17
  KBlock:  Down Facing Skin Angle Hatch   V18
```

```
- New Mnemonic Set -

  BlockL    - Layer id (no vectors)       V19    9/25/87
  BlockLB   - Layer Border                V19
  BlockLH   - Layer Hatch                 V19
  BlockFUF  - Flat Up skin Fill           V19
  BlockNFUB - Near-Flat Up skin Border    V19a   9/28/87
  BlockNFUF - Near-Flat Up skin Fill      V19
  BlockFDF  - Flat Down skin Fill         V19
  BlockNFDB - Near-Flat Down skin Border  V19
  BlockNFDH - Near-Flat Down skin Hatch   V19
  BlockNFDF - Near-Flat Down skin Fill    V19
  BlockFUB  - Near-Flat Down skin Fill    V20    12/4/87
  BlockFDB  - Near-Flat Down skin Fill    V20

-------------------------------------------------------
}

procedure HandleBlockVector(VectorType: Integer);
begin
  case VectorType of
     1: DrawVector;       {ZBlock}
     2: DrawSectionRivet; {XBlock}
     3: DrawSectionRivet; {YBlock}
     4: DrawSectionRivet; {IBlock}
     5: DrawVector;       {SBlock}
     6: DrawVector;       {HBlock}
     7: DrawVector;       {VBlock}
     8: DrawSkinRivet;    {ABlock}
     9: DrawSkinRivet;    {BBlock}
    10: DrawSkinRivet;    {JBlock}
    11: DrawVector;       {CBlock}
    12: DrawVector;       {DBlock}
    13: DrawVector;       {EBlock}
    14: DrawSkinRivet;    {FBlock}
    15: DrawSkinRivet;    {GBlock}
    16: DrawSkinRivet;    {KBlock}
```

```
      20: ;                    {BlockL}
      21: DrawVector;          {BlockLB}
      22: DrawSectionRivet;    {BlockLH}
      23: DrawSkinRivet;       {BlockFUF}
      24: DrawVector;          {BlockNFUB}
      25: DrawSkinRivet;       {BlockNFUF}
      26: DrawSkinRivet;       {BlockFDF}
      27: DrawVector;          {BlockNFDB}
      28: DrawSkinRivet;       {BlockNFDH}
      29: DrawSkinRivet;       {BlockNFDF}
      30: DrawVector;          {BlockFUB}
      31: DrawVector;          {BlockFDB}
    end;
end;

{-------------------------------------------------------}

{
  Break apart input line into two coordinate - (x1,y1)
and
   (x2,y2)
}
procedure GetCoords;
var x_1,y_1,x_2,y_2: Real;
begin
  if VectorFileIsBinary then begin
    if XYScale <>1.000 then begin
      x1 := trunc(Binary_X1 * XYScale);
      y1 := trunc(Binary_Y1 * XYScale);
      x2 := trunc(Binary_X2 * XYScale);
      y2 := trunc(Binary_Y2 * XYScale);
    end else begin
      x1 := trunc(Binary_X1);
      y1 := trunc(Binary_Y1);
      x2 := trunc(Binary_X2);
      y2 := trunc(Binary_Y2);
    end;
```

```
  end else begin  {  ASCII file  }
    BlockIndex := 1;
    UnitVal(x_1,VectorLine,BlockIndex);
    UnitVal(y_1,VectorLine,BlockIndex);
    UnitVal(x_2,VectorLine,BlockIndex);
    UnitVal(y_2,VectorLine,BlockIndex);
    if XYScale<>1.000 then begin
      x1 := trunc(x_1 * XYScale);
      y1 := trunc(y_1 * XYScale);
      x2 := trunc(x_2 * XYScale);
      y2 := trunc(y_2 * XYScale);
    end else begin
      x1 := trunc(x_1);
      y1 := trunc(y_1);
      x2 := trunc(x_2);
      y2 := trunc(y_2);
    end;
  end;
end;

{
  Process block entries until a new layer block header
appears
}
procedure ProcessBlockByLayer;
var
  n,ProcI,Ofs: Integer;
  Ctrl: str10;
  OrgSameBlock,BlockHadVectorsFlag:  Boolean;
  OrgVectorLayer: Word;
  OrgVectorType:  str10;
begin
  { will return at end of block }
  n := 2;
  while (n<length(ControlType)) and not(ControlType[n]
in
   ['0'..'9'])
```

```
    do inc(n);
  ProcI := 0;
  if n=2 then
ProcI:=pos(ControlType[1],BlockIdentifiers);
  if ProcI=0 then begin
    Ctrl := copy(ControlType,1,n-1);
    for j:=1 to BlockMnemonicsCount do
      if Ctrl = BlockMnemonics[j] then ProcI := 19+j;
       {start at 20}
  end;
  if ProcI>0 then begin
    if OpMode=TextOp then write(ControlType,' ')
    else begin
      MoveTo(28*8,13*14);
      OverText(ControlType,5);
      end;
    SetBlockDefaults'
    ProcessCommands'NoQuotes(copy(Control
      Line,ControlIndex,199)));
    VPosition       :=  VectorFileLoc;
    OrgSameBlock    :=  SameBlock;
    OrgVectorLayer  :=  VectorLayer;
    OrgVectorType   :=  VectorType;
    for Ofs:=1 to NumOffsets do begin
      if Ofs>1 then begin
        VectorFileLoc := VPosition;
        ReSeekVectorFile;
      end;
      XOffset     :=  Offsets[Ofs-1,1];
      YOffset     :=  Offsets[Ofs-1,2];
      SameBlock   :=  Or SameBlock;
      VectorLayer :=  Or VectorLayer;
      VectorType  :=  OrgVectorType;
      BlockHadVectorsFlag   := false;
      while SameBlock do begin
        GetVectorEntry;
        if SameBlock then begin
```

```
            BlockHadVectorFlag    := true;
            GetCoords;
            HandleBlockVector(ProcI);
          end;
        end;
        if BlockHadVectorsFlag then LaserExecute;
      end;
    end else
      Message('Unrecognized block type '+VectorType+' in
layer
      '+WordStr(Layer));
end;

procedure ScanAndSendCommands(block: str6);
var
  i,len:  Integer;
begin
  len :=  0;
  i   :=  1;
  while (ParVectorType[i]<>block) and (i<=ParCount)
  do inc(i);
  if i<=ParCount then begin
    ProcessCommands (ParStereoDef[i]);
    inc(len,length(ParStereoDef[i]));
  end;
  i   :=  1;
  while (VectorTypes[i]<>block) and (i<=BlockCount)
  do inc(i);
  if i<=BlockCount then begin
    ProcessCommands (BlockCmds[i]);
    inc(len,length(BlockCmds[i]));
  end;
  if len > 0 then LaserExecute;
end;

{
  Process block entries until a new block entry appears
```

```
}
procedure ProcessBlockByRange;
var
  n,ProcI,Ofs: Integer;
  Ctrl: str10;
  OrgSameBlock,BlockHadVectorsFlag: Boolean;
  OrgVectorLayer: Word;
  OrgVectorType: str10;
begin
  { will return at end of vector block }
  n := 2;
  while (n<length(VectorType)) and not(VectorType[n] in
  ['0'..'9'])
    do inc(n);
  ProcI := 0;
  if n=2 then
ProcI:=pos(VectorType[1],BlockIdentifiers);
  if ProcI=0 then begin
    Ctrl  := copy (VectorType,1,n-1);
    for j :=1 to BlockMnemonicsCount do
      if Ctrl = BlockMnemonics[j] then ProcI  := 19+j;
        { start at 20 }
  end;
  if ProcI>0 then begin
    if OpMode=TextOp then write(VectorType,' ')
    else begin
      MoveTo(28*8,13*14);
      OverText(VectorType,5);
    end;
    ScanAndSendCommands(VectorType);
    VPosition     :=  VectorFileLoc;
    ControlType   :=  VectorType;
    OrgSameBlock  :=  (VectorLayer = ControlLayer) and
      (VectorType = ControlType);
    OrgVectorLayer  := VectorLayer;
    OrgVectorType   := VectorType;
    for Ofs:=1 to NumOffsets do begin
```

```
        if Ofs>1 then begin
          VectorFileLoc :=VPosition;
          ReSeekVectorfile;
        end;
        XOffset   :=  Offsets[Ofs-1,1];
        YOffset   :=  Offsets[Ofs-1,2];
        SameBlock :=  OrgSameBlock;
        VectorLayer := OrgVectorLayer;
        VectorType  := OrgVectorType;
        BlockHadVectorsFlag := false;
        while SameBlock and not EofVectorFile do begin
          GetVectorEntry;
          if SameBlock then begin
            BlockHadVectorsFlag := true;
            GetCoords;
            HandleBlockVector (ProcI);
          end;
        end;
        if BlockHadVectorsFlag then LaserExecute;
      end;
    end else
      Message('Unexpected block type '+VectorType+' in
layer
        '+WordStr(Layer));
end;
{-------------------------------------------------------
  Build Z-table dip routines
}
procedure DelayControl(DelayTime: Integer);
var
  key:  Char;
  ScreenH:  Integer;
  line: String;
  NewZ: Word;
begin
  ScreenH := whereX;
  DeltaSecs := 0;
```

```
  GetTime;
  SaveSec := sec + 60*Minu + 3600*Hour;
  MoveOnFlag  := false;
  SkipFlag  :=  false;
  if Layer >= PauseAfterLayer then begin
    PauseFlag :=  true;
    PauseAfterLayer := 65535;
  and else PauseFlag  := false;
  while (DeltaSecs < DelayTime) and not MoveOnFlag do
begin
    WaitTest;
    if keypressed then begin
      key := upcase(ReadKey);
      case key of
        #0: begin
              key := ReadKey;
              if OpMode = GraphicsOp then begin
                if ord(key) = F1 then SetVisualPage(0)
                else if ord(key) = F2 then SetVisualPage
                 (1);
              end;
            end;
        Ctrl$: key := ReadKey;
        CtrlC: Abort;
        'P': PauseFlag    :=  not PauseFlag;
        'C': begin
              MoveOnFlag  := true; { force exit }
              PauseFlag   := false;
            end;
        'S': begin
              MoveOnFlag  := true;
              SkipFlag    := true;
              PauseFlag   := false;
            end;
        end;
     end;
     if DeltaSecs <> OldDeltaSecs then begin
```

```
        OldDeltaSecs := DeltaSecs;
        temp  := IntStr(DelayTime-DeltaSecs);
        if PauseFlag then temp := temp + '-P'
        else temp := temp + '   ';
        if OpMode=TextOp then begin
          gotoxy(ScreenH,whereY);
          write(temp);
        end else begin
          MoveTo(73*8,7*14);
          OverText(temp,5);
        end;
      end;
    end;  {while}
    gotoxy(ScreenH,whereY);
    if PauseFlag then temp := 'O-P ' else temp := 'O    ';
    if OpMode=TextOp then begin
      gotoxy(ScreenH,whereY);
      write(temp);
    end else begin
      MoveTo(73*8,7*14);
      OverText(temp,5);
    end;
    if PauseFlag then begin { handle pause }
      Action('PAUSED');
      if OpMode=GraphicsOp then begin
        MoveTo(3*8,24*14-2);
        OutText('Building paused.  Press any key to
         continue . . .      ');
  end;
  key := ReadKey; PauseFlag := false;
      if OpMode=GraphicsOp then begin
        MoveTo(3*8,24*14-2);
        OverText('',50);
      end;
    end;
    if (OpMode=TextOp) and (whreX>1) then writeln;
    if SkipFlag then begin { handle layer skip }
```

```
    if OpMode=TextOp then begin
      write('Skip to what Z layer? ');
      readln(line);
    end else begin
      MoveTo(3*8,24*14-2);
      OutText('Skip to what Z layer? ');
      line := GetInput(8);
      MoveTo(3*7,24*(14-2);
      OverText('',30);
      Action ('SKIPPING');
    end;
    if length(line)>0 then begin
      NewZ  := LongVal(line,code);
      if NewZ>0 then StartZ := NewZ;
    end;
  end;
end;

procedure DipAndWait;
var
  clean: Boolean;
begin
  if whereX>1 then writeln;
  Action('DRAWING');
  clean := false;
  repeat
    SendLaserCommand('WI 1000');  { wait for background
laser control to idle }
    if keypressed then HandleKeyboardRequest
      else clean  := true;
  until clean;
  ZMove('MN H+ V' +
RealStr(int(ZTableVeloc*(10000.0)/10000,3)
    + ' A' + RealStr(int(ZTableAccel*10000.0)/10000,3));

  if Demo then begin
    Action('SURFACING');
```

```
    FormatZCommand(-PartHeight);
    DelayControl(OutOfVatDelay);

  if SkipFlag then exit;
    FormatZCommand(PartHeight);
    delay(100);
    WaitForIdleZStage;  {  wait for Z-stage to stop
moving  }
  end;
  Action('DIPPING');
  FormatZCommand(DipDepth);
  if Demo then FormatZCommand(-DipDepth)
  else begin
    if FirstLayer then begin
      FormatZCommand(Thickness-DipDepth);
      FirstLayer  := false;
    end else if ZSpacing <> 0 then
      FormatZCommand((ControlLayer-Layer)/ZSpacing *
        Thickness - DipDepth);
    end

    Action('LEVELING');
    if OpMode=GraphicsOp then begin
      MoveTo(67*8,7*14);
      OutText ('TIME: '+WordStr(RelaxationTime));
      DelayControl(RelaxationTime);
      MoveTo(66*8,7*14);
      OverText('',10);
    end else DelayControl(RelaxationTime);
end;
{-------------------------------------------------------}

procedure HandleHeader(Line: String);
var
  Index: Integer;
begin
  Index  := 2;
```

```
  while (Index<length(Line)) and (Line[Index]='  ')
    do inc(Index);
  UnitVal(SliceRes,Line,Index);
  UnitVal(ZSpacing,Line,Index);
  UnitVal(Thickness,Line,Index);
end;

procedure DetermineNumLayers;
var
  i: Integer;
  range:  LongInt;
begin
  i := 1;
  range :=  ZStart[1];
  LayersToBuild   :=  0;
  while (i <=RangeCount) and (range <=EndZ) do begin
    if range >=StartZ then inc(LayersToBuild);
    if ZSpace[i]>0 then inc(range,ZSpace[i])
      else inc(range);
    if range > ZEnd[i] then begin
      inc(i);
      range   := ZStart[i];
    end;
  end;
end;

procedure DriveByLayers;

begin

{
    determines the next block to execute
    can skip blocks on specific layers
    handles dipping, screen updates
}
  while (ControlLine <> '') and (ControlLayer <= EndZ)
    do begin
```

```
    if ControlLayer < StartZ then begin
      BlockSkip;
      GetLayerControlLine;
    end else if ControlType='#TOP' then begin
      if DipFlag then begin
        DipAndWait;
        DipFlag   := false;
      end;
      MonitorLaserBeamDrift;
      if not SkipFlag then begin
        Layer   := ControlLayer;
        if OpMode=TextOp then write(Layer,':')
        else begin
          MoveTo(14*8,13*14);
          OverText(WordStr(Layer),5);

SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
          ClearViewPort;
          SetViewPort(0,0,GetMaxX,GetMaxY,true);
          MoveTo(7*8,7*14);
          inc(LayerCount);
          if LayersToBuild < 65535 then
            OverText (WordStr(LayerCount)+' OF
              '+WordStr(LayersToBuild),13) else
               OverText(WordStr(LayerCount),13);

        end;
        Action('DRAWING');
        SetBlockDefaults;
        cmd  :=
NoQuotes(copy(ControlLine,ControlIndex,199));
        if length(cmd)>0 then begin
          Process Commands(cmd);
          LaserExecute;
        end;
      end;
      SkipFlag   := false;
```

```
      GetLayerControlLine;
    end else if ControlType='#BTM' then begin
      SetBlockDefaults;
      cmd  :=
NoQuotes(copy(ControlLine,ControlIndex,199));
      if length(cmd)>0 then begin
        ProcessCommands(cmd);
        LaserExecute;
      end;
      DipFlag   := true;
      GetLayerControlLine;
{  will skip blocks for those *.L block entries edited
out  }
    end else begin
      if SameBlock then begin
        ProcessBlockByLayer;
        GetLayerControlLine;
      end else GetVectorEntry;
    end;
  end;
end;

procedure DriveByRanges;
begin
{
    utilizes range information
    can't skip blocks on specific layers
    doesn't support critical areas
    handles dipping, screen updates
  }
  while (not EofVectorFile) and (VectorLayer <= EndZ)
    do begin
    while VectorLayer > RangeZEnd do Read Range;
    if Vectorlayer < StartZ then begin
      BlockSkip;
      DetermineNumLayers; { adjust layer m of n values }
      inc(LayersToBuild,LayerCount);
```

```
    end else begin
      ControlLayer  := VectorLayer; { dip based on
        Control Layer - Layer }
      if DipFlag then begin
        DipAndWait;
        DipFlag  := false;
      end;
      MonitorLaserBeamDrift;
      if VectorLayer >= StartZ then begin
        Layer   :=  ControlLayer;
        if OpMode=TextOp then write(Layer,':')
        else begin
          MoveTo (14*8,13*14);
          OverText(WordStr(Layer),5);

SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
          ClearViewPort;
          SetViewPort(0,0,GetMaxX,GetMaxY,true);
          MoveTo(7*8,7*14);
          inc(LayerCount);
          if LayersToBuild< 65535 then
            OverText(WordStr(LayerCount)+' OF
              '+WordStr(LayersToBuild),13) else
              OverText(WordStr(LayerCount),13);
        end;
        Action('LOADING');
        ScanAndSendCommands('#TOP');
        while (ControlLayer = VectorLayer) and not
          EofVectorFile
        do Process Block By Range;
        ScanAndSendCommands('#BTM');
        DipFlag  := true;
      end;
    end;
  end;
end;
```

```
procedure PartMake;
var
  cmd:  String;
begin
  assign(LastPartFile,LastPartFileName);
  {$I-}
  rewrite(LastPartFile);
  {$I+}
  if IOresult = 0 then begin
    writeln(LastPartFile,PartName);
    close(LastPartFile);
  end;
  if (NumOffsets > 1) and (OpMode = TextOp) then begin
  writeln(IntStr(NumOffsets),' multipls being made
    at once.');
  writeln;
end;
SendLaserCommand('HR;MD BL +');
FirstLayer  := true;
repeat
  LayerCount  :  =  0;
  SendLaserCommand('CL;EC');
  LaserQueueBlockSize  := 0;

{
  read table headers
}
  if LayerCtrl then begin
    repeat
      GetLayerControlLine;
      ifControlLine[1]='*' then
        HandleHeader(ControlLine);
    until ControlLine[1]<>'*';
    LayersToBuild  := NumLayers;
  end else begin
    RangeCount  := 0;
    ControlLine  := 'BEGIN';
```

```
    while ControlLine <>'' do begin { read all ranges in
}
      GetRangeControlLine;
      if ControlLine[1]='*' then
HandleHeader(ControlLine);
      if ControlLine[1] in ['0' .. '9'] then begin
        inc(RangeCount);
        Index  :=  1;
        UnitValInt  (ZStart
[RangeCount],ControlLine,Index);
        UnitValInt  (ZEnd
[RangeCount],ControlLine,Index);
        UnitValShortInt  (ZSpace
[RangeCount],ControlLine,
          Index);
        UnitValShortInt (CureDepth[RangeCount],
          ControlLine,Index);
      end;
    end;
    close(ControlFile);
    DetermineNumLayers;  {determine # of layers from
      range data }
    reset(ControlFile);
    repeat
      GetRangeControlLine;
    until (ControlLine='') or (ControlLine[1]<>'*');
    ReadRange; { load in block data for first range }
    Layer  :=  ZStart[1];
  end;

  BufIndex  := 1;
  if VectorFileIsBinary then VectorFileLoc  := 1
    else VectorFileLoc  := 0;
  BlockVectorCount := 0;
  BlockRead(VectorFile, VectorBuf,VectorRecordSize,
    VectorNumRead);
  repeat
```

```
    GetVectorEntry;
    if VectorLine[1]='*' then HandleHeader(VectorLine);
  until VectorLine[1]<>'*';
  Layer  := -1;

{
  update Action Screen
}
      if OpMode=GraphicsOp then begin
        MoveTo(1*8,7*14);
      if LayersToBuild<65535 then OutText ('LAYER 1
        OF '+WordStr(LayersToBuild))
        else OutText('LAYER 1');
      MoveTo(14*8,12*14);
      OutText(WordStr(StartZ));
      MoveTo(14*8,14*14);
      OutText(WordStr(EndZ));
  end;

    if LayerCtrl then DriveByLayers {decide on control
      routine }
      else DriveByRanges;

    close(ControlFile);
    close(VectorFile);
    if Demo then OpenFiles;
  until not Demo;
end;

{-----------------------------------------------------}

procedure CheckForOptions(var cmd: string);
var
  i,oi,j: Integer;
begin
  cmd  := upper(cmd);
```

```
  i  := pos('/TEXT',cmd); { check for /TEXT text
operation }
  if i>0 then begin
    OpMode  := TextOp;
    delete(cmd,i8,5);
  end;
  i  := pos('/START',cmd); {check for /START Z-layer
start }
  if i>0 then begin
    oi  := i;
    inc(i,6);
    while (i<length(cmd)) and (cmd[i]='  ') do inc(i);
    j  :=  FileStr(i,cmd,'  /');
    val(copy(cmd,i,j-i),StartZ,code);
    delete(cmd,oi,j-oi);
  end;
  i  := pos('/STOP',cmd); { check for /END Z-layer end }
  if i>0 then begin
    oi := i;
    inc(i,5);
    while (i<length(cmd)) and (cmd[i]= ' ') do inc(i);
    j  :=  FindStr(i,cmd,' /');
    val(copy)cmd,i,j-i),EndZ,code);
    delete(cmd,oi,j-oi);
  end;
  i  := pos('/PAUSE',cmd); { check for /PAUSE option}
  if i>0 then begin
    oi := i;
    inc(i,6);
    while (i<length(cmd)) and (cmd[i]= '  ') do inc(i);
    j  := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),PauseAfterLayer,code);
    delete(cmd,oi,j-oi);
  end;
  i  := pos(' /SWAPXY',cmd); {swap X & Y coordinates }
  if i>0 then begin
    SwapXY := true;
```

```
    delete(cmd,i,7);
  end;
  i  := pos(' /X',cmd); { X-axis orientation }
  if i>0 then begin
    NegX  := (cmd[i+2]='-');
    delete(cmd,i,3);
  end;
  i  := pos(' /Y',cmd); {  Y-axis orientation  }
  if i>0 then begin
    NegY  := (cmd[i+2]='-');
    delete(cmd,i,3);
  end;
  i  := pos('/S',cmd);
  if i>0 then begin  { check for /S Show Stuff flag  }
    ShowStuff  := true;
    delete(cmd,i,2);
  end;
  i  := pos('/ZOFF',cmd);  { check for /ZOFF No Z-
Control
    flag  }
  if i>0 then begin
    ZTableOff  := true;
    delete(cmd,i,5);
  end;
  i  := pos('/LOFF',cmd);  {  check for /LOFF No Laser
flag  }
  if i>0 then begin
    LaserOff  := true;
    delete(cmd,i,5);
  end;
  i  := pos('/NEGZ',cmd); {  check for /NEGZ Neg-Z
    Stage flag  }
  if i>0 then begin
    NegZStage  := true;
    delete(cmd,i,5);
  end;
  i  := pos('/HEIGHT',cmd);  {  check for /HEIGHT flag }
```

```
 if i>0 then begin
   oi  := i;
   inc(i,7);
   while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
   j  := FindStr(i,cmd,'/');
   val(copy(cmd,i,j-i),PartHeight,code);
   delete(cmd,oi,j-oi);
 end;
 i  := pos('/DEMO',cmd);  {  check for /DEMO flag  }
 if i>0 then begin
   Demo  := true;
   oi  := i;
   inc(i,5);
   while (i<length(cmd)) and (cmd[i]='  ') do inc(i);
   j  := FindStr(i,cmd,' /');
   val(copy(cmd,i,j-i),OutOfVatDelay,code);
   delete(cmd,oi,j-oi);
 end;
 i  := pos('LC',cmd); { check for Layer Control
   option  }
 if i>0 then begin
   LayerCtrl  := true;
   delete(cmd,i,3);
 end;
 i  := pos(/NODRIFT',cmd);  {  check for no drift
   correction option  }
 if i>0 then begin
   NoDriftFlag  := true;
   delete(cmd,i,8);
 end;
end;

procedure ReadBlockIdFile;
var
  BlockIdFile:  text;
  BlockIdLine:  string;
  Group:  Integer;
```

```
  Block:  String;
begin
  BlockMnemonicsCount  :=  0;
  {$I-}
  assign(BlockIdFile,BlockIdFileName);
  reset(BlockIdFile);
  if IOresult<>0 then begin
    assign(BlockIdFile,HomeDirectory+BlockId
      FileName);
    reset(BlockIdFile);
    if IOresult<>0 then begin
      writeln('Can' 't find Block Id file.');
      halt(2);
    end;
  end;
  {$I+}
  while not eof(BlockIdFile) do begin
    readln(BlockIdFile,BlockIdLine);
    if (length(BlockIdLine)>0) and (BlockIdLine[1]<>'!')

      then begin
      index  := 1;
      UnitValShortInt(Group,BlockIdLine,index);
      UnitStr(Block,BlockIdLine,index);
      if Group=0 then begin
        inc(BlockNmemonicsCount);
        BlockMnemonics[BlockMnemonicsCount]
          := upper(Block);
      end;
    end;
  end;
  close(BlockIdFile);
end;

procedure GetXOffsets(line: string);
begin
  NumXOffsets := 0;
```

```
  repeat
    Offsets[NumXOffsets,1]  := LongVal(line,code);
    inc(NumXOffsets);
    j   := FindStr(1,line,', ');
    if (j=length(line)+1) or (line[j]=' ') then exit;
    line  := copy(line,j+1, 199);
  until NumXOffsets>50;
end;

procedure GetYOffsets(line,string);
begin
  NumYOffsets  := 0;
  repeat
    Offsets[NumYOffsets,2]  := LongVal(line,code);
    inc(NumYOffsets);
    j  := FindStr(1,line,', ');
    if (j=length(line)+1) or (line[j]=' ') then exit;
    line  := copy(line,j+1,199);
  until NumYOffsets>50;
end;
procedure GetXY (line:  String;var Xvar,Yvar:  LongInt);
begin
  Xvar  := LongVal(line,code);
  Yvar  := LongVal(copy(line,code+1,255),code);
end;

procedure ReadBuildParamsFile;
var
  ParamFile:  text;
  count: Integer;
  cmd,ParamLine: String;
begin
  {$I-}
  assign(ParamFile,ParamFileName);
  reset(ParamFile);
  if IOresult<>0 then begin
    assign(ParamFile,'C:\3DSYS\'+ParamFileName);
```

```
    reset(ParamFile);
    if IOresult <>0 then begin
      writeln('Can''t find Build Parameter file.');
      halt(3);
    end;
end;
{$J+}
count  := 0;
while (not eof(ParamFile)) and (count<=8 do begin
    readln(ParamFile, paramLine);
    if (length(ParamLine)>0) and (ParamLine[1]<>'!')
      then begin
    case count of
      0:  begin
            ParamOptions  := upper(NoQuotes(ParamLine));
            if length(ParamOptions)=0 then ParamOptions
              := 'NONE';
            CheckForOptions(ParamLine);
          end;
      1:  ZAddress  :=  IntVal(ParamLine,code);
      2:  ZPitch    :=  RealVal(ParamLine,code);
      3:  XYScale   :=  RealVal(ParamLine,code);
      4:  GetXOffsets(ParamLine);
      5:  GetYOffsets(ParamLine);
      6:  MaxLaserQueueBlockSize  :=
IntVal(ParamLine,code);
      7:  GetXY(ParamLine,GrMinX,GrMinY);
      8:  GetXY(ParamLine,GrMaxX,GrMaxY);
    end;
    inc(count);
  end;
end;
NumOffsets  := NumXOffsets;
if NumOffsets >NumYOffsets then NumOffsets :=
NumYOffsets;
ParCount := 0;
while not eof(ParaFile) do begin
```

```
  cmd := '';
  repeat
    repeat
      if eof(ParamFile) then begin
        SystemError('Error - Unexpected end of BUILD.PRM
          file!');
        halt(4);
      end;
        readln(ParamFile,ParamLine);
        i  := 1; { clean up input, zap some white space
}
        while (i<length(ParamLine)) and (ParamLine[i] in

          [Tab, '  ']) do inc(i);
        delete(ParamLine,1,i-1);
      until (length(ParamLine)>0) and (ParamLine [1]<>
        '!');
      j  := pos('!',ParamLine);
      if j>0 then begin
        dec(j);
        while (j>0) and (ParamLine[j] in [Tab, ' ']) do
        dec(j);
        delete(ParamLine,j+1,199);
      end;
      cmd  := cmd + ParamLine;
    until cmd[length(cmd)]<>';';
    inc(ParCount);
    i  :=  FindStr(1,cmd,',');
    ParVectorType[ParCount]  := copy(cmd,1,i-1);
    ParStereoDef[ParCount]  :=
NoQuotes(copy(cmd,i+1,199));
  end;
  close(ParamFile);
end;

procedure ReadLastPartFile;
begin
```

```
  DefaultPartName  := '';
  assign(LastpartFile,LastPartFileName);
  {$I-}
  reset(LastPartFile);
  {$I+}
  if IOresult <> 0 then exit;
  readln(LastPartFile,DefaultPartName);
  DefaultPartName  :=
upper(copy(DefaultPartName,1,50));
  close(LastPartFile);
end;

procedure ShowFiles(FileSpec: string);
var
  FileCount: Integer;
  DirInfo: SearchRed;
begin
  FileCount  :=  0;
  FindFirst(FileSpec,AnyFile,DirInfo);
  if DosError<>0 then writeln('- none -')
  else begin
    while DosError=0 do begin
      if FileCount>0 then write(spc(11-((whereX-1)
        mod 10+1)));
      inc (FileCount);
      write(copy(DirInfo.Name,1,pos('.',DirInfo.Name)-
1));
      FindNext(DirInfo);
    end;
    writeln;
  end;
end;

procedure GetPartFileNameText;
var
  remX:  Integer;
begin
```

```
  if PartName = '' then begin
    writeln('Directory Listing of Part Files;');
    writeln;
    ShowFiles('*.V');
    writeln;
    write('Part Name:  ');
    remX := whereX;
    write(DefaultPartName);
    gotoxy(remX,whereY);
    readln(PartName);
    if PartName = '' then PartName :+ DefaultPartName;
  end;
  PartName  := upper(PartName);
  CheckForOptions(PartName);
end;

procedure GetPartFileNameGr;
var
  name: str16;
  DirInfo:  SearchRec;
  FileCount,remX:  Integer;
begin
  if length(PartName)=0 then begin
    Filecount := 0;
    MoveTo(1*8,11,*14);
    OutText('Parts Directory Listing:');
    FindFirst('*.V',AnyFile,DirInfo);
    MoveTo(1*8,12*14);
    if DosError<>0 then OutText('- NONE -')
    else begin
      while (DosError = 0) and (GetY < 23*14) do begin
          inc(FileCount);
        name  :=
copy(DirInfo.Name,1,pos('.',DirInfo.Name)-1;
        name  := name + spc(12-length(name));
        OutText(name);
        if GetX >8*60 then MoveTo(1*8,GetY+14;
```

```
      FindNext(DirInfo);
    end;
    writeln;
  end;
  MoveTo(1*8,22*14);
  SetColor(2);
  OutText('Part Name: ');
  SetColor(3);
  remX := GetX;
  OutText(DefaultPartName);
  MoveTo(remX,GetY);
  PartName  := upper(GetInput(5));
  if PartName = '' then PartName := DefaultPartName;
  CheckForOptions(PartName);
 end;
end;

procedure MakeTitle;
begin
  ClearViewPort;
  SetColor(3);
  DrawBorder;
  MoveT(SolidLo(10,5);
  SetTextStyle(TriplexFont,HorizDir,4);
  OutText('3D Systems');
  MoveTo(13,40);
  SetTextStyle(SmallFont,HorizDir,6);
  OutText('Sylmar, California');
  MoveTo (295,9);
  SetTextStyle({SansSerifFont}TriplexFont,HorizDir,4);
  OutText('BUILD");
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo (265,45);
  OutText('Copyright (C) 1988');
  MoveTo(64*8,1*14+7);
  OutText('SLA-1');
  MoveTo(64*8,2*14+7);
```

```
  OutText('Version '+VersionId);
  SetLineStyle(SolidLn,0.ThickWidth);
  Line(0,65,639,65);
  SetLineStyle(SolidLn,0,NormWidth);
end;

procedure MakeParamScreen;
var
  i:  Integer;
begin
  MakeTitle;
  Line(0,6*14+4,639,6*14+4);
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo(1*8,7*14);
  OutText('Options: '+ParamOptions);
  Line(0,8*14+4,639,8*14+4);
  MoveTo(1*8,9*14);
  OutText('XY-Scale:  '+RealStr(XYScale,4));
  MoveTo(55*8,9*14);
  OutText('# Parts:   X  Y');
  Line(0,10*14+4,630,10*14+4);
  Line(68*8-4,8*14+4,68*8-4,23*14+4);
  Line(74*8-4,8*14+4,74*8-4,23*14+4);
  Line(54*8-4,8*14+4,54*8-4,10*14+4);
  Line(0,21*14+4,68*8-4,21*14+4);
  Line(0,23*14+4,639,23*14+4);
  MoveTo(64*8,9*14);
  OutText(WordStr(NumOffsets));
  i  :=  0:
  while (i<12) and (i<NumOffsets) do begin
    MoveTo(68*8,(11+i)*14);
    OutText(LongStr(Offsets[i,1]));
    MoveTo(74*8,(11+i)*14);
    OutText(LongStr(Offsets[i,2]));
    inc(i);
  end;
end;
```

```
procedure MakeAction Screen;
var
  H,V:  Char;
  NegHorz,NegVert:  Boolean;
begin
  MakeTitle;
  Line(0,6*14+4,639,6*14+4);
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo(32*8,7*14);
  OutText('ACTION:  ');
  SetColor(2);
  OutText('INITIALIZING');
  SetColor(3);
  Line(0,8*14+4,639,8*14+4);
  MoveTo(1*8,5*14+3);
  OutText('PART '+PartName);
  MoveTo(64*8,5*14+3);
  OutText('CONTROL: ');
  if LayerCtrl then OutText('LAYER')
    else OutText('RANGE');
  MoveTo(1*8,9*14);
  OutText('START TIME:  '+StartTimeStamp);
  MoveTo(3*8,10*14);
  OutText('END TIME:  '+EndTimeStamp);
  Line(0,11*14+4,35*8+4,11*14+4);
  MoveTo(1*8,12*14);
  OutText('START LAYER:');
  MoveTo(4*8,13*14);
  OutText('ON LAYER:');
  MoveTo(3*8,14*14);
  OutText('END LAYER:');
  Line(0,15*14+4,35*8+4,15*14+4);
  MoveTo(28*8,12*14);
  OutText('BLOCK');
  Line(35*8+4,8*14+4,35*8+4,15*14+4);
  Line(24*8+4,11*14+4,24*8+4,15*14+4);
  Rectangle(45*8+4-1,9*14+4-1,77*8+4+1,22*14+4+1);
```

```
  Line(0,23*14+4,639,23*14+4);
  { draw axes orientation arrows }
  NegVert  := NegY;
  NegHorz  := NegX;
  V  := 'Y';
  H  := 'X';
  if SwapXY then begin
    NegVert  := NegX;
    NegHorz  := NegY;
    V :=  'X';
    H :=  'Y';
  end;
  case NegVert of
    false:  case NegHorz of
              false:  begin

Line(38*8+4,10*14+4,38*8+4,12*14+4);

Line(38*8+4,10*14+4,42*8+4,10*14+4);
                        MoveTo(43*8,10*14);
                        OutText(H);
                        MoveTo(38*8,13*14-4);
                        OutText(V);
                      end;
            true  : begin

Line(42*8+4,10*14+4,42*8+4,12*14+4);

Line(38*8+4,10*14+4,42*8+4,10*14+4);
                        MoveTo(42*8,13*14-4);
                        OutText(V);
                        MoveTo(37*8,10*14);
                        OutText(H);
                      end;
          end;
    true  :  case  NegHorz of
               false:  begin
```

```
                Line(38*8+4,10*14+4,38*8+4,12*14+4);

        Line(38*8+4,12*14+4,42*8+4,12*14+4);
                                MoveTo(38*8,9*14+14);
                                OutText(V);
                                MoveTo(43*8,12*14);
                                OutText(H);
                              end;
                    true   : begin

        Line(42*8+4,10*14+4,42*8+4,12*14+4);

        Line(38*8+4,12*14+4,42*8+4,12*14+4);
                                MoveTo(37*8,12*14);
                                OutText(H);
                                MoveTo(42*8,9*14+4);
                                OutText(V);
                              end;
              end;
          end;
        end;

        {  - - - M A I N - - -  }

        begin
          CheckBreak  :=  false;
          FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);  {
        clear variables  }
          SingleBar  := many(chr(196),45);
          writeln;
          writeln(Sing  Bar);
          writeln('  BUILD, Stereolithography Part-Builder');
          writeln('  Version ', VersionId);
          writeln('  Copyright (c) 1988 by 3D Systems, Inc.');
          writeln('  3D Systems Laser Stereolithography
        System');
          writeln(SingleBar);
```

```
writeln;
OpMode  := TextOp;

check1  :=  LaserPresent;
check2  :=  StereoPresent:
if not (check1 and check2) then halt(1);
OpMode  := GraphicsOp;
StartZ          :=  0;
EndZ            :=  65535;
NumLayers       :=  65535;
PauseAfterLayer :=  65535;
ReadBlockIdFile;
ReadBuildParamsFile;
ReadBeamParamsFile;
ReadLastPartFile;
SetupMonitoring;
ReadCommandLine(cmd);  {  process overriding
  command line  }
i  := pos('/',cmd);
if i=0 then i := length (cmd)+1;
PartName  := upper(copy(cmd,1,i-1));
cmd  := upper(copy(cmd,i,255));
CheckForOptions(cmd);
GrDeltaX  := GrMaxX-GrMinX;
if GrDeltaX < 10 then GrDeltaX  := 10;
GrDeltaY  := GrMaxY-GrMinY;
if GrDeltaY < 10 then GrDeltaY := 10;

if OpMode=TextOp then begin
  GetPartFileNameText;
  if (NumOffsets < 1) or (PartName = '') then
    halt (0);
  writeln;

  OpenFiles
  writeIn('Starting part-making process...');
  GetTimeStamp;
```

```
      StartTimeStamp := CurrentDate+' '+CurrentTime;
      InitZTable;
      PartMake;
      writeln;
      SystemError('Part completed.');
      writeln;
      writeln('Start Time: ',StartTimeStamp);
      GetTimeStamp;
      writeln('  End Time: ',CurrentDate + ' ' +
CurrentTime);
    end else begin
      InstallGraphics;
      BuildImage;
      MakeParamScreen;
      ShowImage;
      GetPartFileNameGr;
      if (NumOffsets < 1) or (PartName = '') then begin
        TextMode(C80);
        halt(0);
      end;
      OpenFiles;
      BuildImage;
      GetTimeStamp;
      StartTimeStamp := CurrentDate + ' ' + CurrentTime;
      MakeActionScreen;
      ShowImage;
      InitZTable;
      PartMake;
      writeln;
      GetTimeStamp;
      EndTimeStamp := CurrentDate + ' ' + CurrentTime;
      MoveTo(13*8,10*14);
      OutText(EndTimeStamp);
      Action('PART COMPLETED');
      repeat
        Ch := ReadKey;
        if (Ch = #0) and keypressed then begin
```

```
            key := ReadKey;  { don't overwrite Ch }
            if OpMode = GraphicOp then begin
              if ord(key) = F1 then SetVisualPage(0)
              else if ord(key) = F2 then SetVisuallPage(1);
            end;
          end;
        until Ch<>#0;
        CloseGraph;
      end;
      Quit;
    end.
```

```
{ ======================================================
LASER.INC
    StereoLithography Laser Controller Software
        Interrupt Interface Routines
    for 3D Systems, Inc.

    Version 2.80  by Tarnz Technologies
======================================================
```

The purpose of these routines is to interface general-purpose Turbo Pascal programs with the memory resident laser controller software.

The machine language-based laser controller software the routines below interface to can be used with other languages like Basic, C, and 8086 Assembly Language.

Turbo Pascal was chosen because of its development environment and low source-to-executable-object code turnaround time. As development progresses and there is no longer a great need for a fast development evironment, these routines should be adapted to C to maintain project software consistency. Turbo C and Microsoft C are the leading choices for C languages.

Below is a list of the routines in this include file. Most of the functions relate directly to specific software interrupt calls. The Code is the related interrupt's AX (AH,AL) value. Some of the interrupt functions require arguments to be passed in other registers, and some will return results. See the actual routines for more details on the interface protocol.

Pascal Function	Code	Description
__LaserOk		see if laser software is installed

__LaserPresent		tell if laser software not installed
__OpenLaser	00 --	open laser DAC channel and initialize
__CloseLaser	01 --	close laser DAC channel and restore system
	02 xx	add command to background command queue
__AddVector	02 00	add vector command
__AddOpenShutter	02 01	add open shutter command
__AddCloseShutter	02 02	add close shutter command
__AddChangeTimebase	02 03	add change time base command
__AddPause	02 04	add pause command
__ResetCmdQueue	03 --	reset background command queue
__GetCmdQueueSize	04 --	get background command queue size
	05 xx	get option
__GetIntNum	05 00	get software interrupt vector number
__GetBkgndTime	05 01	get background task interval time
__GetControlWord	05 02	get laser control word
__AutoShutterModeOn	05 02	get status of auto shutter mode
__PosAckModeOn	05 02	get status of position acknowledge mode
__BlockModeOn	05 02	get status of block mode
__GetStatus	05 03	get laser status
__GetOpenStatus	05 03	get laser channel open status
__GetBkgndIdleStatus	05 03	get background idle status
__GetTurnOffDelay	05 04	get laser turn-off delay
__GetTurnOnDelay	05 05	get laser turn-on delay
__GetHomePosition	05 06	get laser home position
	06 xx	set option

```
__SetIntNum          06 00   set software interrupt
                             vector number
__SetBkgndTime       06 01   set background task interval
                             time
__SetControlWord     06 02   set laser control word
                             (general)
__SetAutoShutMode    06 02   set laser control word, bit0
__SetPosAckMode      06 02   set laser control word, bit1
__SetBlockMode       06 02   set laser control word, bit2
                     06 03   -
__SetTurnOffDelay    06 04   set laser turn-off delay
__SetTurnOnDelay     06 05   set laser turn-on delay
__SetHomePosition    06 06   set laser home position
                     07 xx   background task execution
                             control
__Execute            07 00   execute command queue to
                             current point and stop
__ExecuteClear       07 01   execute command queue to
                             current point, stop, and
                             clear
__Clear              07 02   clear command queue (reset
                             queue's start pointer)
                     08 xx   background task redraw
                             control
__SetStepPeriods     08 00   set multi-pass step periods
__SetRedrawCount     08 01   set redraw pass count
__SetRedrawDelay     08 02   set redraw delay
__Mark               08 03   mark redraw start position
__Redraw             08 04   perform multi-pass redraw
================================================== }

const
_LASERVECTOR     = $64; { Interrupt vector alternative
}
                        { values: $60 through $67.
}
```

```
_PACKET_SIZE     =  13; { length of vector packet
}
_OK              =   0; { Errors returned by Level Zero
}
_INTERNAL_ERROR  =  255;
_MINIMUM_TICKS   =   59; { don't go below 50 us       }
_LASER_QUEUE_MAX = 60000.0; { Laser Queue size        }
__ALLBITS = $FFFF;
__BITVALUE: array[0..15] of Word =
  ($0001, $0002, $0004, $0008, $0010, $0020, $0040,
$0080,
   $0100, $0200, $0400, $0800, $1000, $2000, $4000,
   $8000);

type
  _Str16  = string[16];
  _Str255 = string[255];
  _VectorType = record
      fractX: Byte;   { start coordinates
}
      startX: Word;
      fractY: Byte;     { fractional portion - 1 byte,
                          should be set to 0 }
      startY: Word;     { integer portion  - 2 bytes   }
      deltaX: Integer; { +/- change values in 1/256
units}       deltaY: Integer;
      incr: Word;     { # of increments within vector
                        count}
      end;
  _StepPeriodsType = record
      num    : Byte;
      periods: array[0..9] of Word;
    end;

var
  __IntReg: Registers;
  LaserVersion: _Str16;
```

```
{-------------------------------------------------------

    __LaserOk function
      check to see if laser controller code has been
       installed.

    returns:
    false if code not installed
    true  if code is installed
    LaserVersion contains laser controller version
number

-------------------------------------------------------}

function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with __IntReg do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,__IntReg);
      CopyPtr := Ptr(Es,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __LaserOk := true;
      with __IntReg do
        CopyPtr := Ptr(Es,BX-46);
      move(CopyPtr^[1],LaserVersion[1],18);
      LaserVersion[0] := #4;
    end else begin
      __LaserOk := false;
      LaserVersion := '?.??';
```

```
    end;
  end;

{-------------------------------------------------------

    __LaserPresent function
      same as LaserOk but also prints a message if laser
       system is not installed.

    returns:
    false if code not installed
    true  if code is installed

-------------------------------------------------------}

function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;

{-------------------------------------------------------

    __OpenLaser(Ticks) function
      open up laser DAC channel with the background
server interrupting every Ticks timer 0 ticks.  Input
for timer 0 is a 1.19 MHz clock signal, so if Ticks =
119, the background task would activate once per 100 us;
with Ticks = 1190, once per ms;  etc.

    returns error code

-------------------------------------------------------}
```

```
function __OpenLaser(ticks: Word): Integer;
  begin
{ Min Time CHeck bypassed -- BE CAREFUL!!!!! }
{    if ticks<119 then ticks := 119; }
    with __IntReg do begin
      AX := $0000;
      BX := ticks ;    (* trunc(Period*1.19 {MHz}); *)
      Intr(_LASERVECTOR,__IntReg);
      __OpenLaser := AL;
    end;
  end;

{-------------------------------------------------------

    __CloseLaser function
      close off laser DAC channel after the background
command queue empties.  If laser needs to be stopped
immediately, __ResetVectorPool should be called before
this routine.

    returns error code

-------------------------------------------------------}

function __CloseLaser: Integer;
  begin
    with __IntReg do begin
      AX := $0100;
      Intr(_LASERVECTOR,__IntReg);
      __CloseLaser := AL;
    end;
  end;

{-------------------------------------------------------

    __AddVector(startX,startY,deltaX,deltaY,incr)
function
```

```
      add the vector to the background command queue.
      Vector start coordinates are (startX,startY) and
      their change value is (deltaX,deltaY).  The number
      of incremental steps in the vector is incr.
deltaX
      and deltaY are in a special format (high byte
      is +/- integer portion; low byte is 1/256 units)
      conducive to high-speed DAC updating.  See the
3DPC
      protocol specification for more information.
      returns error code

  ------------------------------------------------- }

function __AddVector(startX,startY: Word;deltaX,deltaY:
Integer;
                     incr: Word): Integer;
  var
    VectorPacket: _VectorType;
  begin
    if incr>0 then begin
      VectorPacket.fractX := 0;       { set up vector
packet }
      VectorPacket.startX := startX;
      VectorPacket.fractY := 0;
      VectorPacket.startY := startY;
      VectorPacket.deltaX := deltaX;
      VectorPacket.deltaY := deltaY;
      VectorPacket.incr   := incr;
      with __IntReg do begin
        AX := $0200;
        ES := Seg(VectorPacket);     { point ES:BX to
vector packet }
        BX := Ofs(VectorPacket);
        Intr(_LASERVECTOR,__IntReg);
        __AddVector := AL;
      end;
```

```
    end else __AddVector := 0;
  end;

{
-------------------------------------------------------

    __AddOpenShutter function
      add the open shutter command to the background
command queue.  The shutter will be opened by the
background task when it gets to this command in the
command queue.

    returns error code

  ------------------------------------------------------ }

function  __AddOpenShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0201;
      Intr(_LASERVECTOR,__IntReg);
      __AddOpenShutter := AL;
    end;
  end;

{ ------------------------------------------------------

    __AddCloseShutter function
      add the close shutter command to the background
command queue.  The shutter will be closed by the
background task when it gets to this command in the
command queue.

    returns error code

  ------------------------------------------------------ }
```

```
function __AddCloseShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0202;
      Intr(_LASERVECTOR,__IntReg);
      __AddCloseShutter := AL;
    end;
  end;

{ ------------------------------------------------

    __AddChangeTimebase(NewTicks) function
      add the change time base command to the background
command queue.
      The timer that awakens the badckground task will
be reloaded with the count of NewTicks by the background
task when it gets to this command in the command queue.

    returns error code

  ------------------------------------------------ }

function __AddChangeTimebase(NewTicks: Word): Integer;
  begin with __IntReg do begin
      AX := $0203;
      BX := NewTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddChangeTimebase := AL;
    end;
  end;

{
------------------------------------------------------

    __AddPause(DelayTicks) function
      add the pause command to the background command
queue.  When the background task gets to this command,
```

```
it will pause for DelayTicks number of timer ticks.  If
the Background Interrupt Time is 119
      (100 us), then a pause of 1 ms will ensue if
DelayTicks is 10.
      Afterwards, background activity resumes.

    returns error code

  ------------------------------------------------- }

function __AddPause(DelayTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0204;
      BX := DelayTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddPause := AL;
    end;
  end;

{
-------------------------------------------------------

    __ResetCmdQueue function
      clear out the background command queue.  Usually
used as part of an abort procedure.  Best when used just
before a __CloseLaser call.

    returns error code

  ------------------------------------------------- }

function __ResetCmdQueue: Integer;
  begin
    with __IntReg do begin
      AX := $0300;
      Intr(_LASERVECTOR,__IntReg);
```

```
      __ResetCmdQueue := AL;
    end;
  end;

{
-------------------------------------------------------

    __GetCmdQueueSize function
      get the background command queue's size in actual
number of bytes.
      Number of vectors is variable but can be estimated
by the number returned by this routine / 13.

    returns error code

  ------------------------------------------------------ }

function __GetCmdQueueSize(var size: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0400;
      Intr(_LASERVECTOR,__IntReg);
      size := BX;
      if size<0 then size := size + 65536;
      __GetCmdQueueSize := AL;
    end;
  end;

{
-------------------------------------------------------

    __GetIntNum(num) function
      place in num the software interrupt number.  Not
really important unless we have a software interrupt
conflict.

    returns error code
```

```
  ------------------------------------------------- }

function __GetIntNum(var num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0500;
      Intr(_LASERVECTOR,__IntReg);
      num := BL;
      __GetIntNum := AL;
    end;
  end;

{-------------------------------------------------------

    __GetBkgndTime(ticks) function
      place in time the background task interrupt
interval time in actual 1.19 MHz tick counts.  (119 is
100 us, 1190 is 1 ms, etc.)

    returns error code

  ------------------------------------------------- }

function __GetBkgndTime(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0501;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetBkgndTime := AL;
    end;
  end;

{-------------------------------------------------------

    __GetControlWord(ctrl) function
```

```
      place in ctrl the Laser Control Word showing the
status of various hardware and software mechanisms.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          0     shutter control, 0=open/1=closed

    returns error code

 ------------------------------------------------------ }

function __GetControlWord(var ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0502;
      Intr(_LASERVECTOR,__IntReg);
      ctrl := BX;
      __GetControlWord := AL;
    end;
  end;

{-------------------------------------------------------

    __AutoShutterModeOn function
      returns the status of the Auto Shutter Mode (true
for on).

 ------------------------------------------------------ }

function __AutoShutterModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
```

```
  begin
    Er := __GetControlWord(ctrl);
    __AutoShutterModeOn := (ctrl and __BITVALUE[0]) > 0;
  end;

{-------------------------------------------------------

    __PosAckModeOn function
      returns the status of the Position Acknowledge
Mode (true for on).

 ------------------------------------------------------- }

function __PosAckModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __PosAckModeOn := (ctrl and __BITVALUE[1]) > 0;
  end;

{-------------------------------------------------------

    __BlockModeOn function
      returns the status of the Block Mode (true for
on).

  ------------------------------------------------------- }

function __BlockModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __BlockModeOn := (ctrl and __BITVALUE[2]) > 0;
```

```
  end;

{-------------------------------------------------------

    __GetStatus(status1,status2) function
      place in status1 and status2 the Laser system's
stati of various non-programmable hardware and software
mechanisms.

      word bit      description
      --------      ---------------------------------
      status1.0     laser system channel open status
      status1.1     laser system software interrupt
                    established (sic)
      status1.2     laser system background task in idle
                    state
      status1.3-15  -
      status2.0     bit 0 of Port C1, X and Y-axis
                    position acknowledge
      status2.1     bit 1 of Port C1, Z-axis position
                    acknowledge
      status2.2     bit 2 of Port C1
      status2.3     bit 3 of Port C1
      status2.4-15  -

    returns error code

  ------------------------------------------------------ }

function __GetStatus(var status1,status2: Word):
Integer;
  begin
    with __IntReg do begin
      AX := $0503;
      Intr(_LASERVECTOR,__IntReg);
      status1 := BX;
      status2 := CX;
```

```
      __GetStatus := AL;
    end;
  end;

{-------------------------------------------------------

    __GetOpenStatus function
      tells whether or not the laser system
communications channel has been opened and the timer
tick trap installed.  The function returns a boolean
value of TRUE if the channel is open.

    returns boolean value

 ------------------------------------------------------- }

function __GetOpenStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetOpenStatus := (stat1 and
__BITVALUE[0]) > 0
      else __GetOpenStatus := false;
  end;

{ -----------------------------------------------

    __GetBkgndIdleStatus function
      tells whether or not the laser system background
task is currently in the idle state where no more laser
activity can be performed without some more
foreground-sourced commands.  The function returns a
boolean value of TRUE if the background task is idle.

    returns boolean value
```

```
 ------------------------------------------------- }

function __GetBkgndIdleStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetBkgndIdleStatus := (stat1 and
__BITVALUE[2]) > 0
      else __GetBkgndIdleStatus := false;
  end;

{-------------------------------------------------
    __GetTurnOffDelay(ticks) function
      get the idle state's laser turn-off delay in
interrupt tick counts.

    returns error code

 ------------------------------------------------- }

function __GetTurnOffDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0504;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOffDelay := AL;
    end;
  end;

{ -------------------------------------------------

    __GetTurnOnDelay(ticks) function
      get the idle state's laser turn-on delay in
interrupt tick counts.
```

```
    returns error code

  ------------------------------------------------- }

function __GetTurnOnDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0505;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOnDelay := AL;
    end;
  end;

{ -------------------------------------------------

    __GetHomePosition(X,Y) function
      get the idle state's laser home position.

    returns error code


-------------------------------------------------}

function __GetHomePosition(var X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0506;
      Intr(_LASERVECTOR,__IntReg);
      X := BX;
      Y := CX;
      __GetHomePosition := AL;
    end;
  end;

{-------------------------------------------------
```

```
    __SetIntNum(num) function
      set the software interrupt number to num.  All
software interrupt call after this point must use this
new interrupt number.  The interrupt vector for the old
number is restored.

    returns error code


----------------------------------------------------}

function __SetIntNum(num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0600;
      BL := num;
      Intr(_LASERVECTOR,__IntReg);
      __SetIntNum := AL;
    end;
  end;

{----------------------------------------------------

    __SetBkgndTime(ticks) function
      set the background task interrupt interval to
ticks which has the actual number of 1.19 MHz tick
counts.  (119 is 100 us, etc.)

    returns error code


----------------------------------------------------}

function __SetBkgndTime(ticks: Word): Integer;
  begin
```

```
    if ticks<_MINIMUM_TICKS then ticks :=
_MINIMUM_TICKS;
    with __IntReg do begin
      AX := $0601;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetBkgndTime := AL;
    end;
  end;

{ ------------------------------------------------------

    __SetControlWord(ctrl) function
      set the Laser Control Word to ctrl.  It is
recommended that most software programs use the
__AlterControlWordBit and the specific mode set routines
which follow.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          4     shutter control, 0=open/1=closed

    returns error code

------------------------------------------------------}

function __SetControlWord(ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0602;
      BX := ctrl;
      Intr(_LASERVECTOR,__IntReg);
      __SetControlWord := AL;
```

```
    end;
  end;

{-------------------------------------------------------

    __AlterControlWordBit(op,bit) function
      alter the state of the indicated Control Word bit.
If op is 'S' or
      '+', the bit is set;  if op is 'C' or '-', the bit
is cleared.

    returns error code


-------------------------------------------------------}

function __AlterControlWordBit(op: Char;bit: Word):
Integer;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    if Er=0 then begin
      if upcase(op) in ['S','+'] then ctrl := ctrl or
__BITVALUE[bit]
        else if upcase(op) in ['C','-'] then
          ctrl := ctrl and (__ALLBITS-__BITVALUE[bit]);
      Er := __SetControlWord(ctrl);
    end;
    __AlterControlWordBit := Er;
  end;
```

```
{-------------------------------------------------------

    __SetAutoShutterMode(state) function
      set the state ('S'or '+' to set, 'C' or '-' to
clear) of the
      Auto-Shutter Mode.

    returns error code


-------------------------------------------------------

function __SetAutoShutterMode(op: Char): Integer;
  begin
    __SetAutoShutterMode := __AlterControlWordBit(op,0);
  end;

{-------------------------------------------------------

    __SetPosAckMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the
      Position-Acknowledge Mode.

    returns error code


-------------------------------------------------------}

function __SetPosAckMode(op: Char): Integer;
  begin
    __SetPosAckMode := __AlterControlWordBit(op,1);
  end;
```

```
{-------------------------------------------------------

    __SetBlockMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the
      Block Mode where the 07 commands (below) are
active.

    returns error code


-------------------------------------------------------}

function __SetBlockMode(op: Char): Integer;
  begin
    __SetBlockMode := __AlterControlWordBit(op,2);
  end;

{-------------------------------------------------------

    __SetTurnOffDelay(ticks) function
      sets the idle state's laser turn-off delay to
ticks interrupt counts.

    returns error code


-------------------------------------------------------}

function __SetTurnOffDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0604;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOffDelay := AL;
    end;
```

```
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end
  val(line,num,code);
  if code<>0 then val (copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  ReadVal := num;
end;

{
  Pick out a number from a string starting at Index and
  going to the next space or comma or end-of-string
}
procedure UnitValShortInt(var Storage: Integer;Text:
  String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure Unit ValInt(var Storage: LongInt;
  Text:String;var Index:Integer);
var
  i:integer;
begin
  Storage: = LongVal(Copy(Text,Index, 99),i);
  if i = 0 then Index: = length(Text)+1
    else inc(Index,i);
  end;

procedure UnitVal(var Storage:Real;Text:String;var
  Index:  Integer);
var
```

```
      BX := X;
      CX := Y;
      Intr(_LASERVECTOR,__IntReg);
      __SetHomePosition := AL;
    end;
  end;

{-------------------------------------------------------

    __Execute function
      set the background task running.  The background
task will process those commands added to the command
queue from the last Clear Operation up to this __Execute
call.   For the first __Execute, then background task
will start at the beginning of the queue.  New commands
can be added to the command queue while the background
task is running.

      When the background task is finished processing
the current command block, it will automatically shut
down, placing the laser beam at the preset home
position.

      To make programs written for earlier versions
(where the background task free-ran) compatible,
__Execute, __ExecuteClear, and __Clear calls (see below)
should be used wherever appropriate.

    returns error code


-------------------------------------------------------}

function __Execute: Integer;
  begin
    with __IntReg do begin
      AX := $0700;
```

```
      Intr(_LASERVECTOR,__IntReg);
      __Execute := AL;
    end;
  end;

{-------------------------------------------------------

    __ExecuteClear function
      perform the equivalent of the __Execute function
(see above) and a
      __Clear function (see below) function.  The
__ExecuteClear function
      provides a convenient shorthand for:

        Er := __Execute;
        Er := __Clear;

    returns error code


-------------------------------------------------------}

function __ExecuteClear: Integer;
  begin
    with __IntReg do begin
      AX := $0701;
      Intr(_LASERVECTOR,__IntReg);
      __ExecuteClear := AL;
    end;
  end;

{-------------------------------------------------------

    __Clear function
      clear the command queue.  Future __Execute and
__ExecuteClear calls
```

```
      will start the background task processing command
queue entries added after the call to this routine.  An
Execute Operation asked for immediately after a Clear
Operation has no effect.

      If __Clear is not called between consecutive
Execute Operations, then background task will reprocess
the command queue entries for the first Execute
Operation.  Sometimes this is desired, and sometimes it
isn't.

    returns error code

-------------------------------------------------------}

function __Clear: Integer;
  begin
    with __IntReg do begin
      AX := $0702;
      Intr(_LASERVECTOR,__IntReg);
      __Clear := AL;
    end;
  end;

{-------------------------------------------------------

    __SetStepPeriods function
      set the step periods for the multiple passes.
array contains number of step periods to define (1 to
10) and the step period values in 0.82 us (1.19 Mhz)
units.

    returns error code

-------------------------------------------------------}
```

```
function __SetStepPeriods(var sp_array:
_StepPeriodsType): Integer;
  begin
    with __IntReg do begin
      AX := $0800;
      ES := Seg(sp_array);     { point ES:BX to step
periods array }
      BX := Ofs(sp_array);
      Intr(_LASERVECTOR,__IntReg);
      __SetStepPeriods := AL;
    end;
  end;

{-------------------------------------------------------

    __SetRedrawCount function
      set the number of passes for the background task's
redraw (1-8).

    returns error code


-------------------------------------------------------}

function __SetRedrawCount(count: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0801;
      if count<1 then count := 1
        else if count>10 then count := 10;
      BX := count;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawCount := AL;
    end;
  end;

{-------------------------------------------------------
```

```
    __SetRedrawDelay function
      set the number of timer ticks the background task
should delay before starting a redraw pass.

    returns error code


-------------------------------------------------------}

function __SetRedrawDelay(delay: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0802;
      BX := delay;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __Mark function
      marks the current position as the start of the
next redraw.
      Usually only necessary at the beginning of a
block.

    returns error code


-------------------------------------------------------}

function __Mark: Integer;
  begin
    with __IntReg do begin
      AX := $0803;
```

```
      Intr(_LASERVECTOR,__IntReg);
      __Mark := AL;
    end;
  end;

{-------------------------------------------------------

    __Redraw function
      causes the background task to perform a multi-pass
redraw operation when it is encountered in the
Background Command Queue.  __Redraw also marks so as to
minimize use of commands.

    returns error code


-------------------------------------------------------}

function __Redraw: Integer;
  begin
    with __IntReg do begin
      AX := $0804;
      Intr(_LASERVECTOR,__IntReg);
      __Redraw := AL;
    end;
  end;

{ end of LASER.INC }^Z{
=======================================================
    StereoLithography Laser Controller Software
        Interrupt Interface Routines
    for 3D Systems, Inc.

    Version 2.80  by Tarnz Technologies
=======================================================
```

The purpose of these routines is to interface general-purpose Turbo Pascal programs with the memory resident laser controller software.

The machine language-based laser controller software the routines below interface to can be used with other languages like Basic, C, and 8086 Assembly Language.

Turbo Pascal was chosen because of its development environment and low source-to-executable-object code turnaround time. As development progresses and there is no longer a great need for a fast development evironment, these routines should be adapted to C to maintain project software consistency. Turbo C and Microsoft C are the leading choices for C languages.

Below is a list of the routines in this include file. Most of the functions relate directly to specific software interrupt calls. The Code is the related interrupt's AX (AH,AL) value. Some of the interrupt functions require arguments to be passed in other registers, and some will return results. See the actual routines for more details on the interface protocol.

Pascal Function	Code	Description
__LaserOk		see if laser software is installed
__LaserPresent		tell if laser software not installed
__OpenLaser	00 --	open laser DAC channel and initialize
__CloseLaser	01 --	close laser DAC channel and restore system
	02 xx	add command to background command queue
__AddVector	02 00	add vector command

__AddOpenShutter	02	01	add open shutter command
__AddCloseShutter	02	02	add close shutter command
__AddChangeTimebase	02	03	add change time base command
__AddPause	02	04	add pause command
__ResetCmdQueue	03	--	reset background command queue
__GetCmdQueueSize	04	--	get background command queue size
	05	xx	get option
__GetIntNum	05	00	get software interrupt vector number
__GetBkgndTime	05	01	get background task interval time
__GetControlWord	05	02	get laser control word
__AutoShutterModeOn	05	02	get status of auto shutter mode
__PosAckModeOn	05	02	get status of position acknowledge mode
__BlockModeOn	05	02	get status of block mode
__GetStatus	05	03	get laser status
__GetOpenStatus	05	03	get laser channel open status
__GetBkgndIdleStatus	05	03	get background idle status
__GetTurnOffDelay	05	04	get laser turn-off delay
__GetTurnOnDelay	05	05	get laser turn-on delay
__GetHomePosition	05	06	get laser home position
	06	xx	set option
__SetIntNum	06	00	set software interrupt vector number
__SetBkgndTime	06	01	set background task interval time

```
__SetControlWord     06 02   set laser control word
                             (general)
__SetAutoShutMode    06 02   set laser control word,
                             bit 0
__SetPosAckMode      06 02   set laser control word,
                             bit 1
__SetBlockMode       06 02   set laser control word,
                             bit 2
                     06 03   -
__SetTurnOffDelay    06 04   set laser turn-off delay
__SetTurnOnDelay     06 05   set laser turn-on delay
__SetHomePosition    06 06   set laser home position
                     07 xx   background task execution
                             control
__Execute            07 00   execute command queue to
                             current point and stop
__ExecuteClear       07 01   execute command queue to
                             current point, stop, and
                             clear
__Clear              07 02   clear command queue (reset
                             queue's start pointer)
                     08 xx   background task redraw
                             control
__SetStepPeriods     08 00   set multi-pass step periods
__SetRedrawCount     08 01   set redraw pass count
__SetRedrawDelay     08 02   set redraw delay
__Mark               08 03   mark redraw start position
__Redraw             08 04   perform multi-pass redraw
================================================== }

const
_LASERVECTOR     = $64; { Interrupt vector alternative
}
                        { values: $60 through $67.
}
_PACKET_SIZE     =  13; { length of vector packet
}
```

```
_OK              =   0; { Errors returned by Level Zero
}
_INTERNAL_ERROR  =  255;
_MINIMUM_TICKS   =   59; { don't go below 50 us       }
_LASER_QUEUE_MAX = 60000.0; { Laser Queue size        }
__ALLBITS = $FFFF;
__BITVALUE: array[0..15] of Word =
  ($0001, $0002, $0004, $0008, $0010, $0020, $0040,
$0080,
   $0100, $0200, $0400, $0800, $1000, $2000, $4000,
   $8000);

type
  _Str16  = string[16];
  _Str255 = string[255];
  _VectorType = record
      fractX: Byte;   { start coordinates
}
      startX: Word;
      fractY: Byte;     { fractional portion - 1 byte,
                          should be set to 0 }
      startY: Word;     { integer portion  - 2 bytes   }
      deltaX: Integer; { +/- change values in 1/256
units}        deltaY: Integer;
      incr: Word;     { # of increments within vector
                        count}
      end;
  _StepPeriodsType = record
      num    : Byte;
      periods: array[0..9] of Word;
    end;

var
  __IntReg: Registers;
  LaserVersion: _Str16;

{ ------------------------------------------------------
```

```
    __LaserOk function
      check to see if laser controller code has been
       installed.

    returns:
    false if code not installed
    true  if code is installed
    LaserVersion contains laser controller version
number

  ------------------------------------------------
}

function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with __IntReg do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,__IntReg);
      CopyPtr := Ptr(Es,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __LaserOk := true;
      with __IntReg do
        CopyPtr := Ptr(Es,BX-46);
      move(CopyPtr^[1],LaserVersion[1],18);
      LaserVersion[0] := #4;
    end else begin
      __LaserOk := false;
      LaserVersion := '?.??';
```

```
    end;
  end;

{
-------------------------------------------------------

    __LaserPresent function
      same as LaserOk but also prints a message if laser
       system is not installed.

    returns:
    false if code not installed
    true  if code is installed

  ------------------------------------------------------ }

function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;

{ ------------------------------------------------------

    __OpenLaser(Ticks) function
      open up laser DAC channel with the background
server interrupting every Ticks timer 0 ticks.  Input
for timer 0 is a 1.19 MHz clock signal, so if Ticks =
119, the background task would activate once per 100 us;
with Ticks = 1190, once per ms;  etc.

    returns error code

 ------------------------------------------------------ }
```

```
function __OpenLaser(ticks: Word): Integer;
  begin
{ Min Time CHeck bypassed -- BE CAREFUL!!!!! }
{    if ticks<119 then ticks := 119; }
    with __IntReg do begin
      AX := $0000;
      BX := ticks ;     (* trunc(Period*1.19 {MHz}); *)
      Intr(_LASERVECTOR,__IntReg);
      __OpenLaser := AL;
    end;
  end;

{ ------------------------------------------------------

    __CloseLaser function
      close off laser DAC channel after the background
command queue empties.  If laser needs to be stopped
immediately, __ResetVectorPool should be called before
this routine.

    returns error code

  ------------------------------------------------------ }

function __CloseLaser: Integer;
  begin
    with __IntReg do begin
      AX := $0100;
      Intr(_LASERVECTOR,__IntReg);
      __CloseLaser := AL;
    end;
  end;

{ ------------------------------------------------------
```

```
    __AddVector(startX,startY,deltaX,deltaY,incr)
function
      add the vector to the background command queue.
      Vector start coordinates are (startX,startY) and
      their change value is (deltaX,deltaY).  The number
      of incremental steps in the vector is incr.
deltaX
      and deltaY are in a special format (high byte
      is +/- integer portion; low byte is 1/256 units)
      conducive to high-speed DAC updating.  See the
3DPC
      protocol specification for more information.
      returns error code

  ------------------------------------------------- }

function __AddVector(startX,startY: Word;deltaX,deltaY:
Integer;
                     incr: Word): Integer;
  var
    VectorPacket: _VectorType;
  begin
    if incr>0 then begin
      VectorPacket.fractX := 0;       { set up vector
                                        packet }
      VectorPacket.startX := startX;
      VectorPacket.fractY := 0;
      VectorPacket.startY := startY;
      VectorPacket.deltaX := deltaX;
      VectorPacket.deltaY := deltaY;
      VectorPacket.incr   := incr;
      with __IntReg do begin
        AX := $0200;
        ES := Seg(VectorPacket);     { point ES:BX to
                                       vector packet }
        BX := Ofs(VectorPacket);
        Intr(_LASERVECTOR,__IntReg);
```

```
        __AddVector := AL;
      end;
    end else __AddVector := 0;
  end;

{ ------------------------------------------------------

    __AddOpenShutter function
      add the open shutter command to the background
command queue.  The shutter will be opened by the
background task when it gets to this command in the
command queue.

    returns error code

  ------------------------------------------------------ }

function  __AddOpenShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0201;
      Intr(_LASERVECTOR,__IntReg);
      __AddOpenShutter := AL;
    end;
  end;

{ ------------------------------------------------------

    __AddCloseShutter function
      add the close shutter command to the background
command queue.  The shutter will be closed by the
background task when it gets to this command in the
command queue.

    returns error code

  ------------------------------------------------------ }
```

```
function __AddCloseShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0202;
      Intr(_LASERVECTOR,__IntReg);
      __AddCloseShutter := AL;
    end;
  end;

{ ------------------------------------------------

    __AddChangeTimebase(NewTicks) function
      add the change time base command to the background
command queue.
      The timer that awakens the badckground task will
be reloaded with the count of NewTicks by the background
task when it gets to this command in the command queue.

    returns error code

  ------------------------------------------------ }

function __AddChangeTimebase(NewTicks: Word): Integer;
  begin with __IntReg do begin
      AX := $0203;
      BX := NewTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddChangeTimebase := AL;
    end;
  end;

{ ------------------------------------------------
```

```
    __AddPause(DelayTicks) function
      add the pause command to the background command
queue.  When the background task gets to this command,
it will pause for DelayTicks number of timer ticks.  If
the Background Interrupt Time is 119
      (100 us), then a pause of 1 ms will ensue if
DelayTicks is 10.
      Afterwards, background activity resumes.

    returns error code

  ------------------------------------------------- }

function __AddPause(DelayTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0204;
      BX := DelayTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddPause := AL;
    end;
  end;

{ -------------------------------------------------

    __ResetCmdQueue function
      clear out the background command queue.  Usually
used as part of an abort procedure.  Best when used just
before a __CloseLaser call.

    returns error code

  ------------------------------------------------- }

function __ResetCmdQueue: Integer;
  begin
    with __IntReg do begin
```

```
      AX := $0300;
      Intr(_LASERVECTOR,__IntReg);
      __ResetCmdQueue := AL;
    end;
  end;

{ ------------------------------------------------------

    __GetCmdQueueSize function
      get the background command queue's size in actual
number of bytes.
      Number of vectors is variable but can be estimated
by the number returned by this routine / 13.

    returns error code

  ------------------------------------------------------ }

function __GetCmdQueueSize(var size: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0400;
      Intr(_LASERVECTOR,__IntReg);
      size := BX;
      if size<0 then size := size + 65536;
      __GetCmdQueueSize := AL;
    end;
  end;

{ ------------------------------------------------------

    __GetIntNum(num) function
      place in num the software interrupt number.  Not
really important unless we have a software interrupt
conflict.

    returns error code
```

```
  ------------------------------------------------
}

function __GetIntNum(var num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0500;
      Intr(_LASERVECTOR,__IntReg);
      num := BL;
      __GetIntNum := AL;
    end;
  end;

{ ------------------------------------------------

    __GetBkgndTime(ticks) function
      place in time the background task interrupt
interval time in actual 1.19 MHz tick counts.  (119 is
100 us, 1190 is 1 ms, etc.)

    returns error code

  ------------------------------------------------ }

function __GetBkgndTime(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0501;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetBkgndTime := AL;
    end;
  end;

{ ------------------------------------------------
```

```
    __GetControlWord(ctrl) function
      place in ctrl the Laser Control Word showing the
status of various hardware and software mechanisms.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          0     shutter control, 0=open/1=closed

    returns error code

  ------------------------------------------------- }

function __GetControlWord(var ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0502;
      Intr(_LASERVECTOR,__IntReg);
      ctrl := BX;
      __GetControlWord := AL;
    end;
  end;

{ -------------------------------------------------

    __AutoShutterModeOn function
      returns the status of the Auto Shutter Mode (true
for on).

  ------------------------------------------------- }

function __AutoShutterModeOn: Boolean;
  var
    Er: Integer;
```

```
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __AutoShutterModeOn := (ctrl and __BITVALUE[0]) > 0;
  end;

{ ------------------------------------------------------

    __PosAckModeOn function
      returns the status of the Position Acknowledge
Mode (true for on).

  ------------------------------------------------------ }

function __PosAckModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __PosAckModeOn := (ctrl and __BITVALUE[1]) > 0;
  end;

{ ------------------------------------------------------

    __BlockModeOn function
      returns the status of the Block Mode (true for
on).

  ------------------------------------------------------ }

function __BlockModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
```

```
    __BlockModeOn := (ctrl and __BITVALUE[2]) > 0;
  end;

{-------------------------------------------------------

    __GetStatus(status1,status2) function
      place in status1 and status2 the Laser system's
stati of various non-programmable hardware and software
mechanisms.

      word bit      description
      --------      ---------------------------------
      status1.0     laser system channel open status
      status1.1     laser system software interrupt
                    established (sic)
      status1.2     laser system background task in idle
                    state
      status1.3-15  -
      status2.0     bit 0 of Port C1, X and Y-axis
                    position acknowledge
      status2.1     bit 1 of Port C1, Z-axis position
                    acknowledge
      status2.2     bit 2 of Port C1
      status2.3     bit 3 of Port C1
      status2.4-15  -

    returns error code

  ------------------------------------------------- }

function __GetStatus(var status1,status2: Word):
Integer;
  begin
    with __IntReg do begin
      AX := $0503;
      Intr(_LASERVECTOR,__IntReg);
      status1 := BX;
```

```
      status2 := CX;
      __GetStatus := AL;
    end;
  end;

{ ------------------------------------------------------

    __GetOpenStatus function
      tells whether or not the laser system
communications channel has been opened and the timer
tick trap installed.  The function returns a boolean
value of TRUE if the channel is open.

    returns boolean value

 ------------------------------------------------------ }

function __GetOpenStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetOpenStatus := (stat1 and
__BITVALUE[0]) > 0
      else __GetOpenStatus := false;
  end;

{ ------------------------------------------------

    __GetBkgndIdleStatus function
      tells whether or not the laser system background
task is currently in the idle state where no more laser
activity can be performed without some more
foreground-sourced commands.  The function returns a
boolean value of TRUE if the background task is idle.
```

```
    returns boolean value

 ------------------------------------------------------ }

function __GetBkgndIdleStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetBkgndIdleStatus := (stat1 and
__BITVALUE[2]) > 0
      else __GetBkgndIdleStatus := false;
  end;

{------------------------------------------------------
    __GetTurnOffDelay(ticks) function
      get the idle state's laser turn-off delay in
interrupt tick counts.

    returns error code

 ------------------------------------------------------ }

function __GetTurnOffDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0504;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOffDelay := AL;
    end;
  end;
```

```
{ ------------------------------------------------------

    __GetTurnOnDelay(ticks) function
      get the idle state's laser turn-on delay in
interrupt tick counts.

    returns error code

  ------------------------------------------------------ }

function __GetTurnOnDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0505;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOnDelay := AL;
    end;
  end;

{ ------------------------------------------------------

    __GetHomePosition(X,Y) function
      get the idle state's laser home position.

    returns error code


------------------------------------------------------}

function __GetHomePosition(var X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0506;
      Intr(_LASERVECTOR,__IntReg);
      X := BX;
      Y := CX;
```

```
      __GetHomePosition := AL;
    end;
  end;

{-------------------------------------------------------

    __SetIntNum(num) function
      set the software interrupt number to num.  All
software interrupt call after this point must use this
new interrupt number.  The interrupt vector for the old
number is restored.

    returns error code


-------------------------------------------------------}

function __SetIntNum(num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0600;
      BL := num;
      Intr(_LASERVECTOR,__IntReg);
      __SetIntNum := AL;
    end;
  end;

{ -----------------------------------------------------

    __SetBkgndTime(ticks) function
      set the background task interrupt interval to
ticks which has the actual number of 1.19 MHz tick
counts.  (119 is 100 us, etc.)

    returns error code
```

```
-------------------------------------------------------}

function __SetBkgndTime(ticks: Word): Integer;
  begin
    if ticks<_MINIMUM_TICKS then ticks :=
_MINIMUM_TICKS;
    with __IntReg do begin
      AX := $0601;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetBkgndTime := AL;
    end;
  end;

{ -------------------------------------------------------

    __SetControlWord(ctrl) function
      set the Laser Control Word to ctrl.  It is
recommended that most software programs use the
__AlterControlWordBit and the specific mode set routines
which follow.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          4     shutter control, 0=open/1=closed

    returns error code

-------------------------------------------------------}

function __SetControlWord(ctrl: Word): Integer;
  begin
```

```
    with __IntReg do begin
      AX := $0602;
      BX := ctrl;
      Intr(_LASERVECTOR,__IntReg);
      __SetControlWord := AL;
    end;
  end;

{-------------------------------------------------------

    __AlterControlWordBit(op,bit) function
      alter the state of the indicated Control Word bit.
If op is 'S' or
      '+', the bit is set;  if op is 'C' or '-', the bit
is cleared.

    returns error code


-------------------------------------------------------}

function __AlterControlWordBit(op: Char;bit: Word):
Integer;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    if Er=0 then begin
      if upcase(op) in ['S','+'] then ctrl := ctrl or
__BITVALUE[bit]
        else if upcase(op) in ['C','-'] then
          ctrl := ctrl and (__ALLBITS-__BITVALUE[bit]);
      Er := __SetControlWord(ctrl);
    end;
    __AlterControlWordBit := Er;
  end;
```

```
{-------------------------------------------------------

    __SetAutoShutterMode(state) function
      set the state ('S'or '+' to set, 'C' or '-' to
clear) of the
      Auto-Shutter Mode.

    returns error code


-------------------------------------------------------

function __SetAutoShutterMode(op: Char): Integer;
  begin
    __SetAutoShutterMode := __AlterControlWordBit(op,0);
  end;

{-------------------------------------------------------

    __SetPosAckMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the
      Position-Acknowledge Mode.

    returns error code


-------------------------------------------------------}

function __SetPosAckMode(op: Char): Integer;
  begin
    __SetPosAckMode := __AlterControlWordBit(op,1);
  end;

{-------------------------------------------------------
```

```
    __SetBlockMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the
      Block Mode where the 07 commands (below) are
active.

    returns error code


-------------------------------------------------------}

function __SetBlockMode(op: Char): Integer;
  begin
    __SetBlockMode := __AlterControlWordBit(op,2);
  end;

{-------------------------------------------------------

    __SetTurnOffDelay(ticks) function
      sets the idle state's laser turn-off delay to
ticks interrupt counts.

    returns error code


-------------------------------------------------------}

function __SetTurnOffDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0604;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOffDelay := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __SetTurnOnDelay(ticks) function
      sets the idle state's laser turn-on delay to ticks
interrupt counts.

    returns error code


-------------------------------------------------------}

function __SetTurnOnDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0605;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOnDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __SetHomePosition(X,Y) function
      sets the idle state's laser home position.

    returns error code


-------------------------------------------------------}

function __SetHomePosition(X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0606;
      BX := X;
      CX := Y;
```

```
      Intr(_LASERVECTOR,__IntReg);
      __SetHomePosition := AL;
    end;
  end;

{-------------------------------------------------------

    __Execute function
      set the background task running.  The background
task will process those commands added to the command
queue from the last Clear Operation up to this __Execute
call.   For the first __Execute, then background task
will start at the beginning of the queue.  New commands
can be added to the command queue while the background
task is running.

      When the background task is finished processing
the current command block, it will automatically shut
down, placing the laser beam at the preset home
position.

      To make programs written for earlier versions
(where the background task free-ran) compatible,
__Execute, __ExecuteClear, and __Clear calls (see below)
should be used wherever appropriate.

    returns error code


-------------------------------------------------------}

function __Execute: Integer;
  begin
    with __IntReg do begin
      AX := $0700;
      Intr(_LASERVECTOR,__IntReg);
      __Execute := AL;
```

```
   end;
  end;

{ ------------------------------------------------------

    __ExecuteClear function
      perform the equivalent of the __Execute function
(see above) and a
      __Clear function (see below) function.  The
__ExecuteClear function
      provides a convenient shorthand for:

        Er := __Execute;
        Er := __Clear;

    returns error code


------------------------------------------------------}

function __ExecuteClear: Integer;
  begin
    with __IntReg do begin
      AX := $0701;
      Intr(_LASERVECTOR,__IntReg);
      __ExecuteClear := AL;
    end;
  end;

{------------------------------------------------------

    __Clear function
      clear the command queue.  Future __Execute and
__ExecuteClear calls
      will start the background task processing command
queue entries added after the call to this routine.  An
```

```
Execute Operation asked for immediately after a Clear
Operation has no effect.

      If __Clear is not called between consecutive
Execute Operations, then background task will reprocess
the command queue entries for the first Execute
Operation.  Sometimes this is desired, and sometimes it
isn't.

    returns error code


-------------------------------------------------------}

function __Clear: Integer;
  begin
    with __IntReg do begin
      AX := $0702;
      Intr(_LASERVECTOR,__IntReg);
      __Clear := AL;
    end;
  end;

{-------------------------------------------------------

    __SetStepPeriods function
      set the step periods for the multiple passes.
array contains number of step periods to define (1 to
10) and the step period values in 0.82 us (1.19 Mhz)
units.

    returns error code


-------------------------------------------------------}
```

```
function __SetStepPeriods(var sp_array:
_StepPeriodsType): Integer;
  begin
    with __IntReg do begin
      AX := $0800;
      ES := Seg(sp_array);     { point ES:BX to step
periods array }
      BX := Ofs(sp_array);
      Intr(_LASERVECTOR,__IntReg);
      __SetStepPeriods := AL;
    end;
  end;

{-------------------------------------------------------

    __SetRedrawCount function
      set the number of passes for the background task's
redraw (1-8).

    returns error code


-------------------------------------------------------}

function __SetRedrawCount(count: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0801;
      if count<1 then count := 1
        else if count>10 then count := 10;
      BX := count;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawCount := AL;
    end;
  end;

{-------------------------------------------------------
```

```
    __SetRedrawDelay function
      set the number of timer ticks the background task
should delay before starting a redraw pass.

    returns error code


-------------------------------------------------------}

function __SetRedrawDelay(delay: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0802;
      BX := delay;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __Mark function
      marks the current position as the start of the
next redraw.
      Usually only necessary at the beginning of a
block.

    returns error code


-------------------------------------------------------}

function __Mark: Integer;
  begin
    with __IntReg do begin
      AX := $0803;
```

```
      Intr(_LASERVECTOR,__IntReg);
      __Mark := AL;
    end;
  end;

{-------------------------------------------------------

    __Redraw function
      causes the background task to perform a multi-pass
redraw operation when it is encountered in the
Background Command Queue.  __Redraw also marks so as to
minimize use of commands.

    returns error code


-------------------------------------------------------}

function __Redraw: Integer;
  begin
    with __IntReg do begin
      AX := $0804;
      Intr(_LASERVECTOR,__IntReg);
      __Redraw := AL;
    end;
  end;

{ end of LASER.INC }^Z
```

```
{-------------------------------------------------
LCTRL.INC.
                Laser Controller Routines
-------------------------------------------------}

{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
    if LaserOk then LaserPresent := true
```

```
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{
  Check for Memory-Resident STEREO Stereolithography
  Device Driver
}
function StereoOk: Boolean;
  type
    CopyType = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;

function StereoPresent: Boolean;
```

```
  begin
    if StereoOk then StereoPresent := true
    else begin
      StereoPresent := false;
      SystemError('STEREO driver not installed.');
    end;
  end;

procedure ShowError(Error: Integer);
begin
  case Error of
    0: ;
    1: PrintError('1. Laser interface card not found');
    2: PrintError('2. Laser channel not open');
    3: PrintError('3. Background Command Queue full
     timeout');
    4: PrintError('4. Invalid interrupt time');
    5: PrintError('5. Not in block mode');
    6: PrintError('6. Exec Queue limit timeout');
    7: PrintError('127. Bad SIH command parameter');
    8: PrintError('128. Unrecognized STEREO command');
    255: PrintError('255. Internal error (please
     report)');
  else PrintError(IntStr(Error)+'. Unrecognized SIH
     error');
  end;
end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - -

 Pass Command on to Memory Resident STEREO Device Driver

- - - - - - - - - - - - - - - - - - - - - - - - - - - - }

function STEREO(LaserCommand: String): Integer;
begin
  STEREO := 0;
```

```
  if length(LaserCommand)>0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      writeln('   Laser>',LaserCommand);
    end;
    if not LaserOff then begin
      if LaserCommand[1]<>'!' then with regs do begin
        LaserCommand := LaserCommand + chr(0);
        AX := $00;   { ASCII text format }
        BX := Ofs(LaserCommand)+1; { pass pointer to
         string }
        ES := Seg(LaserCommand);
        Intr(STEREOVECTOR,regs);
        STEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if keypressed then HandleKeyboardRequest;
end;

function BinarySTEREO(CommandBuffer: Pointer): Integer;
var
  BufSeg,BufOfs: Word;
begin
  BinarySTEREO := 0;
  BufSeg := seg(CommandBuffer^);
  BufOfs := ofs(CommandBuffer^);
  if Mem[BufSeg:BufOfs] > 0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      write('   BinaryLaser>');
      for i:=0 to 11 do
write(HexByte(Mem[BufSeg:BufOfs+i]),' ');
      writeln;
    end;
    if not LaserOff then begin
```

```
        with regs do begin
          AX := $01;   { binary format }
          BX := BufOfs; { pass pointer to binary command }
          ES := BufSeg;
          Intr(STEREOVECTOR,regs);
          BinarySTEREO := AX and $FF;
          QSize := CX;
        end;
      end;
    end;
    if keypressed then HandleKeyboardRequest;
  end;
```

```
{-------------------------------------------------------
GRAPH.INC
                General Graphics Routines
-------------------------------------------------------}

procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn,0,ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure BuildImage;
begin
  OnScreen := 1-OnScreen;
  SetActivePage(OnScreen);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x + len * TextWidth('A'),y + TextHeight(Text)
+ 1);
  SetColor(color)
  OutTextXY(x,y,Text);
end;

procedure ShowImage;
begin
  SetVisualPage(OnScreen);
```

```
end;

function GetInput(maxlen: Integer): String;
var
  key: Char;
  temp: String;
  First: Boolean;
  baseX,baseY,index: Integer;
begin
  temp := '';
  index := 0;
  baseX := getX;
  baseY := getY;
  key := chr(0);
  SetVisualPage(OnScreen);
  First := true;
  while key <> Cr do begin
    MoveTo(baseX+index*8,baseY);
    SetColor(2);
    if not First then OverText('_',51-index) { show
      cursor }
    else begin
      OutText('_');
      First := false;
    end;
    SetColor(3);
    key := ReadKey;
    if key = #0 then begin
      if keypressed then key := ReadKey;
      key := #0;
    end;
    if key = CtrlC then Abort;
    MoveTo(baseX+index*8,baseY);
    if key in [Bs,Cr,' '..'~'] then begin
      if key = Bs then begin
        if index > 0 then dec(index);
      end else if key in[' '..'~'] then begin
```

```
        OverText(key,1);
        if index < maxlen then inc(index);
        temp[index] := key;
      end;
    end;
  end;
  MoveTo(baseX+index*8,baseY);
  SetColor(0);
  OutText('_');   { turn off cursor }
  SetColor(3);
  temp[0] := chr(index);
  GetInput := temp;
  SetColor(3);
end;

procedure GrAbort(Msg: string);
begin
  writeln(Msg,': ',GraphErrorMsg(GraphResult));
  halt(1);
end;

procedure InstallGraphics;
begin
  { Register all the drivers }
  if RegisterBGIdriver(@CGADriverProc) < 0 then
    GrAbort('CGA');
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    GrAbort('EGA/VGA');

  { Register all the fonts }
  if RegisterBGIfont(@SansSerifFontProc) < 0 then
    GrAbort('SansSerif');
  if RegisterBGIfont(@SmallFontProc) < 0 then
    GrAbort('Small');
  if RegisterBGIfont(@TriplexFontProc) < 0 then
    GrAbort('Triplex');
```

```
  GraphDriver := Detect;      { autodetect the hardware }
  InitGraph(GraphDriver,GraphMode,'');activate graphics }
  if GraphResult<>grOk then               { any errors? }
  begin
    Writeln('Graphics init error: ', GraphErrorMsg
      (GraphDriver));
    Halt(1);
  end;
  SetVisualPage(1);
  ClearDevice;
  OnScreen := 1;   { start with Screen 0 }
end;

{
  Print action information
}
procedure Action(Text: String);
var
  i: Integer;
begin
  if OpMode=TextOp then write(Text,' ')
  else begin
    MoveTo(40*8,7*14);
    SetColor(2);
    OverText(Text,20);
    SetColor(3);
  end;
end;

procedure Message(Text: String);
begin
  MoveTo(3*8,24*14-2);
  OverText(Text,length(Text)+5;
end;
```

```
{-------------------------------------------------------
KEYBRD.INC
                    Keyboard Routines
-------------------------------------------------------}

procedure Abort;
var
  key: Char;
begin
  if OpMode=GraphicsOp then TextMode(C80);
  while keypressed do key := ReadKey;
  halt(127);
end;

function STEREO(LaserCommand: String): Integer; forward;
procedure ShowError(Error: Integer);            forward;

procedure Quit;
var
  key: Char;
begin
  ShowError(STEREO('SP 100;EC;EC'));
  if OpMode=GraphicsOp then TextMode(C80);
  halt(0);
end;

procedure HandleKeyboardRequest;
var
  key: Char;
begin
  while keypressed do begin
    key := ReadKey;
    if key = CtrlS then key := ReadKey;
    if key = CtrlC then Abort;
    if (key = #0) and keypressed then begin
      key := ReadKey;
      if OpMode = GraphicsOp then begin
```

```
            if ord(key) = F1 then SetVisualPage(0)
            else if ord(key) = F2 then SetVisualPage(1);
          end;
        end;
      end;
    end;
```

```
{-------------------------------------------------------
TIME.INC
                    Time/Date Routines
-------------------------------------------------------}

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Compute day-of-week using Harry's modified Zeller
    Algorithm
}
function Zeller(month,day,year: Integer): Integer;
begin
  if month<3 then dec(year);
```

```
  Zeller := (Zel[month] + day + year + (year shr 2)) mod
7;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
    standard
    PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A; { wait for update to finish (then
      there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00; { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02; { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;   { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;   { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;   { get month }
              Mon := port[$71];
```

```
                port[$70] := $0A;
                if (port[$71] and $80)=0 then begin
                  port[$70] := $09;   { get year }
                  Year := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then good :=
                   true;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
    Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
      from BCD }
    Minu := (Minu shr 4) * 10 + (Minu and $0F);
    Hour := (Hour shr 4) * 10 + (Hour and $0F);
    Day  := (Day  shr 4) * 10 + (Day  and $0F);
    Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
    Year := (Year shr 4) * 10 + (Year and $0F);
    WeekDay := Zeller(Mon,Day,Year);
  end;

  {
    Get a time stamp consisting of the relative day and
    the current time
  }
  procedure GetTimeStamp;
  var
    tdow,tday,tmon,tyear,tsec,tminu,thour: str10;
    AmPm: Char;
  begin
    GetTime;
    tmon  := PadSp(Mon);
    tday  := LZero(Day);
    tyear := LZero(Year);
```

```
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPmt'M';
end;

{
  Compute difference in seconds for a calling wait
routine
}
procedure WaitTest;
begin
  GetTime;
  NewSec := Sec + 60*Minu + 3600*Hour;
  DeltaSecs := NewSec - SaveSec;
  if DeltaSecs < 0 then DeltaSecs := DeltaSecs + 86400;
end;
```

```
{-------------------------------------------------------
UTILITY
                    Utility Routines
-------------------------------------------------------}

procedure beep;
begin
  Sound(1000);
  delay(300);
  NoSound;
end;

function sgn(num: Real): Real;
begin
  if num<0 then sgn := -1
  else if num>0 then sgn := 1
  else sgn := 0;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
    upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp := temp+Ch;
```

```
  many := temp;
end;

function min(int1,int2: Integer): Integer;
begin
  if int1<int2 then min := int1
    else min := int2;
end;

function max(int1,int2: Integer): Integer;
begin
  if int1>int2 then max := int1
    else max := int2;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;
function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
  Str(num,temp);
  LongStr := temp;
end;
```

```
function RealStr(num: Real;AfterDecimal: Integer):
str80;
var
  i: Integer;
  temp: str80;
begin
  Str(num:10:AfterDecimal,temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc := copy(sp80+sp80,1,len);
end;

{
  convert number to hex byte string
}
function HexByte(num: Byte): str4;
const
  HexChars: str16 = '0123456789ABCDEF';
begin
  HexByte := HexChars[num shr 4 + 1] + HexChars[num
    and $0F + 1];
end;

{
  read the DOS command line
}
procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
```

```
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: str80): str80;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Remove all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;

{
  Find any char of Chars in Main starting at index Start
    Returns length(Main)+1 if no character of Chars in
    Main
}
function FindStr(Start: Integer;Main: string;Chars:
  str10): Integer;
var
  i,j,min: Integer;
```

```
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
  for i := 1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function WordVal(line: String;var code: Integer): Word;
var
  code2: Integer;
  num: Word;
begin
  val(line,num,code);
  if code <>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;
function LongVal(line: String;var code: Integer):
LongInt;
var
```

```
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
var
  code2: Integer;
  num: Real;
begin
  if line[1]='.' then line := '0' + line;
  val(line,num,code);
  if code <>0 then val(copy(line,1,code-1),num,code2);
  RealVal := num;
end;

{
  Pick out a number from a string starting at Index and
    going to the next space or comma or end-of-string
}
procedure UnitValShortInt(var Storage: Integer;Text:
  String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else Index := Index + i;
end;

procedure UnitVal(var Storage: LongInt;Text: String;var
  Index: Integer);
var
  i: Integer;
```

```
begin
  Storage := RealVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else Index := Index + i;
end;

{
  Pick out text from a string starting at Index and
  going
    to the next comma or end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
  Integer);
var
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  if OpMode = TextOp then writeln(Text)
    else begin
      MoveTo(3*8,24*14-2);
      OverText(Text,50);
      Ch := ReadKey;
      TextMode(C80);
    end;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'SUPER: ',Text);
  close(ErrorFile);
```

```
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{
  IEEE / Turbo Pascal floating point number conversions
    }
procedure ConvertRtoS(r,S: FpPointerType);
  { Converts a Real type (6 byte) number to a Single (4
     byte) }
var
  e: Byte;
begin
  e:=r^[0];
  if e<>0 then Dec(e,2);
  s^[3]:=(r^[5] and $80) or ((e shr 1) and $7F);
  s^[2]:=((e shl 7) or (r^[5] and $7F)) and $FF;
  s^[1]:=r^[4];
  s^[0]:=r^[3];
end;

procedure ConvertStoR(S,r: FpPointerType);
  { Converts a Single type (4 byte) number to a Real
    (6 byte) }
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
```

```
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;
```

```
{-------------------------------------------------------
ZSTAGE.INC
                Z-Stage Elevator Routines
-------------------------------------------------------}

procedure InitZTable;
begin
  port[ZAddress+1] := $64;
  port[ZAddress+1] := $60;
  delay(100);
  port[ZAddress+1] := $40;
  port[ZAddress+1] := $60;
  delay(100);
end;

procedure ZTable(Ch: Char);
var
  b: Integer;
begin
  repeat
    b := port[ZAddress+1];
  until (b and $10)>0;
  port[ZAddress] := ord(Ch);
  port[ZAddress+1] := $70;
  repeat
    B:=port[ZAddress+1];
  until (b and $10)=0;
  port[ZAddress+1] := $60;
end;

{
  Send command to Z-axis stepper controller
}
procedure ZMove(Cmd: string);
begin
  if ShowStuff then begin
    if whereX>1 then writeln;
```

```
    writeln('   ZTable>',Cmd);
  end;
  if not ZTableOff then begin
    Cmd := Cmd + Cr;
    for i:=1 to length(Cmd) do ZTable(Cmd[i]);
  end;
end;

procedure FormatZCommand(Distance: Real);
begin
  Distance := sgn(Distance) * int(abs(Distance) * ZPitch
    * 1000 / 25.4);
  if NegZStage then Distance := -Distance;
  ZMove('D'+RealStr(Distance,0)+' G');
end;

procedure WaitForIdleZStage;
var
  b: Word;
begin
  repeat
    b := port[ZAddress+1];
  until keypressed or (b and 2>0);
end;
^Z
```

CALOUT.BAS

```
'CALOUT.BAS        3/28/88
ver$ = "ver 3.00"
'to convert sensor readings to calibration table
'specifies directory for STEREO.GEO, accepts .CAL input

'readings are every 889 bits = 0.2500"
'nearest above 65536 = 74 x 889  =  65786  =  65535 +
251
'therefore our table goes from 0 to 74
RangeError = 0
tlo%=0
thi%=74
DIM x3#(tlo%:thi%,tlo%:thi%),y3#(tlo%:thi%,tlo%:thi%)
  'X & Y calibration tables
DIM
x1#(tlo%:thi%+1),y1#(tlo%:thi%+1),x2#(tlo%:thi%+1),y2#
  (tlo%:thi%+1) 'sensors # 1,2

CLS
?
?"  CALOUT ";ver$
?
reread:

INPUT "Exact name of calibration file to be adjusted
";nm$
InputFile$= nm$                '#1 - rename when saving
i%= INSTR(1,nm$,".")
IF i%>0 THEN nm$=LEFT$(nm$,i%-1)
OutputFile$=nm$+".GEO"         '#2
DriftFile$= nm$+".DRF"         '#2

'nearest to center is 37 x 889 = 32893  =  32767 +
  126  (.0354")
'and our readings (-6" to + 6") are from array
  locations 13 to 61
```

```
'(NOTE X Axis is reversed: this was corrected by
  reading X from 6 to -6)

clo%=13
chi%=61
inc#=889

CLS
KEY OFF
LOCATE 5,1,1
?"Program to convert data from Beam Profiler
  Calibration Run"
?
INPUT "Customer Name and/or SLA-1 Serial # ",nm$
?
INPUT "DX Serial # ",dx$
?
INPUT "Date of Calibration ",dt$

CLS
LOCATE 5,1,0
?"Filling Uncorrected Table"
'---------------------------
FOR i% = tlo% TO thi%
   x3#=inc#*i%
   IF x3#<5 THEN x3#=5
   IF x3#>65530 THEN x3#=65530
   FOR j% = tlo% TO thi%
      x3#(i%,j%)=x3#
   NEXT j%
NEXT i%
FOR j% = tlo% TO thi%
   y3#=inc#*j%
   IF y3#<5 THEN y3#=5
   IF y3#>65530 THEN y3#=65530
   FOR i% = tlo% TO thi%
      y3#(i%,j%)=y3#
```

```
   NEXT i%
NEXT j%


Reading:
CLS
LOCATE 5,1,0
?"Reading Measured Values"
'------------------------------
' may later have text in .CAL file which should be saved

SUB ReadLine(x3#,y3#)     'later use temperature to
  check for drift
    LOCAL a$, i%
   ReadLine:
    LINE INPUT #1,a$
    IF LEFT$(a$,1)="!" THEN ReadLine
    i%=INSTR(a$,",")                    'find first comma
    x3#=VAL(LEFT$(a$,i%))               'pick up x value
    a$=RIGHT$(a$,LEN(a$)-i%)            'remove used
portion
                                         of a$
    i%=INSTR(a$,",")                    'get next comma
    y3#=VAL(LEFT$(a$,i%))               'pickup up y value
END SUB

SUB ReadFour(x3#,y3#,x4#,y4#)
    LOCAL a$, i%
  ReadBack:
    LINE INPUT #1,a$
    IF LEFT$(a$,1)="!" THEN ReadBack

    i%=INSTR(a$,",")                    'find first comma
    x3#=VAL(LEFT$(a$,i%))               'pick up x value

    a$=RIGHT$(a$,LEN(a$)-i%)            'remove used
portion
```

```
                                                of a$
    i%=INSTR(a$,",")                           'get next comma
    y3#=VAL(LEFT$(a$,i%))                      'pickup up y value

    a$=RIGHT$(a$,LEN(a$)-i%)                   'remove used
portion
                                               of a$
    i%=INSTR(a$,",")                           'find next comma
    x4#=VAL(LEFT$(a$,i%))                      'pick up x value

    a$=RIGHT$(a$,LEN(a$)-i%)                   'remove used
portion
                                               of a$
    i%=INSTR(a$,",")                           'get next comma
    y4#=VAL(LEFT$(a$,i%))                      'pickup up y value

END SUB

OPEN INPUTFILE$ FOR INPUT AS #1

'Should save the 32767,32767 readings at beginning & end
'to check for relative movement, or uncompensated drift.

FOR i% = clo% TO chi%
   CALL ReadFour(x1#(i%),y1#(i%),x2#(i%),y2#(i%))
   FOR j% = clo% TO chi%
      CALL ReadLine(x3#(i%,j%),y3#(i%,j%))
      IF x3#(i%,j%)<10000 OR x3#(i%,j%)>55000 THEN
        PRINT "WARNING X Value Out of Range. X = ",
          x3#(i%,j%)
        RangeError = -1
      END IF
      IF y3#(i%,j%)<10000 OR y3#(i%,j%)>55000 THEN
        PRINT "WARNING Y Value Out of Range. Y = ",
          x3#(i%,j%)
        RangeError = -1
      END IF
```

```
   NEXT j%
NEXT I%
i%=chi%+1                'just in case BASIC changes
CALL ReadFour(x1#(i%),y1#(i%),x2#(i%),y2#(i%))
        ' may add 32767,32767 at beginning & end  CLOSE
#1

IF RangeError <> 0 Then
  PRINT "Pres Enter to Continue...";
  Input Temp$
END IF
CLS
LOCATE 5,1,0
?"Finding Mirror Drift"
'-------------------------------------------------

'  ***  NOTE we have re-grouped x1,x2 y1,y2 here, we
read
  in x1,y1 x2,y2

FOR I%=clo% TO chi%+1
   x1tot#=x1tot#+x1#(i%)/(chi%+1-clo%+1)    'could use
    arrays, but much easier to make errors!
   x2tot#=x2tot#+x2#(i%)/(chi%+1-clo%+1)
   y1tot#=y1tot#+y1#(i%)/(chi%+1-clo%+1)
   y2tot#=y2tot#+y2#(i%)/(chi%+1-clo%+1)
NEXT I%

FOR I%=clo% TO chi%+1
   xd#=x1tot#-x1#(i%)
    if xdmax#<xd# then xdmax#=xd#
    if xdmin#>xd# then xdmin#=xd#
    x1dev#=x1dev#+xd#*xd#/(chi%+1-clo%+1)
   xd#=x2tot#-x2#(i%)
    if xdmax#<xd# then xdmax#=xd#
    if xdmin#>xd# then xdmin#=xd#
    x2dev#=x2dev#+xd#*xd#/(chi%+1-clo%+1)
```

```
   yd#=y1tot#-y1#(i%)
    if ydmax#<yd# then ydmax#=yd#
    if ydmin#>yd# then ydmin#=yd#
    y1dev#=y1dev#+yd#*yd#/(chi%+1-clo%+1)
   yd#=y2tot#-y2#(i%)
    if ydmax#<yd# then ydmax#=yd#
    if ydmin#>yd# then ydmin#=yd#
    y2dev#=y2dev#+yd#*yd#/(chi%+1-clo%+1)
NEXT I%

OPEN DriftFile$ FOR OUTPUT AS #2

PRINT #2,"!CALOUT ";ver$;" Drift Report File"
PRINT #2,"!SLA-1 ";nm$;", DX serial # ";dx$;" on ";dt$

PRINT #2,"!Position values used for calibration, & RMS
  deviations (bits)"
PRINT #2,"!Sensor1 ";
PRINT #2, USING
"######.##";x1tot#,y1tot#,SQR(x1dev#),SQR(y1dev#)
PRINT #2,"!Sensor2 ";
PRINT #2, USING
"######.##";x2tot#,y2tot#,SQR(x2dev#),SQR(y2dev#)

PRINT #2,"   x1tot#   x2tot#   y1tot#   y2tot# (bits)"
PRINT #2, USING "######.##";x1tot#,x2tot#,y1tot#,y2tot#
PRINT #2,"   x1dev#   x2dev#   y1dev#   y2dev# (bits
RMS)"
PRINT #2, USING
"######.##";SQR(x1dev#),SQR(x2dev#),SQR(y1dev#),SQR
  (y2dev#)
PRINT #2,""
PRINT #2,"   x1#(i%)  x2#(i%)  y1#(i%)  y2#(i%)  (bits)"
FOR i%=clo% TO chi%+1
   PRINT #2, USING
"######.##";x1#(i%)-x1tot#;x2#(i%)-x2tot#;_
```

```
y1#(i%)-y1tot#;y2#(i%)-y2tot#
NEXT i%
CLOSE #2

FOR I%=clo% TO chi%+1        're-work the drift
  information ------------------
  xof#=((x1#(i%)+x2#(i%))-(x1tot#+x2tot#))/2  'bits
  yof#=((y1#(i%)+y2#(i%))-(y1tot#+y2tot#))/2  'bits
  xgn#=(x1#(i%)-x2#(i%))/(x1tot#-x2tot#)  'approximately
    = 1
  ygn#=(y1#(i%)-y2#(i%))/(y1tot#-y2tot#)  'approximately
    = 1
  're-use the x1#() etc. arrays
   x1#(i%)=xof#
   x2#(i%)=xgn#
   y1#(i%)=yof#
   y2#(i%)=ygn#
NEXT i%
OPEN DriftFile$ FOR APPEND AS #2
PRINT #2,""
PRINT #2,"   X offset X gain   Y offset Y gain (bits,
  bits/10";CHR$(34);")"
PRINT #2,""
FOR i%=clo% TO chi%+1
   PRINT #2, USING
"#######.#";x1#(i%);(x2#(i%)-1)*35560_
                                y1#(i%);(y2#(i%)-1)*35560
NEXT i%
CLOSE #2

'needs to show temp. drift & ask whether to correct or
use
  un-corrected * * *
    if ydmax#<yd# then ydmax#=yd#
    if ydmin#>yd# then ydmin#=yd#
   xd#=xdmax#-xdmin#  'both started from zero, must be
    + & -
```

```
   yd#=ydmax#-ydmin#
   IF xd#<yd# THEN xd#=yd#  'select larger
   PRINT
   PRINT "From measuring the apparent position of the
    beam profile sensors"
   PRINT "during calibration, we find total effective
    mirror drift"
   PRINT USING "####";xd#/3.556;
   PRINT " mils "
   PRINT
   FOR i = 1 to 10000
   NEXT i

   IF xd#>500 THEN
      PRINT "Program found wrong profiler pinholes, "
      PRINT ""
      PRINT "File will be written without attempting to
        correct for drift"
      GOTO writeout
   END IF
   IF xd#>35 THEN
      PRINT "Excessive drift, you may want to run again
"
      PRINT "                ?temperature drift
      PRINT "                ?air conditioning direct
air
                             currents
      PRINT "                ?insufficient warmup"
      PRINT
      IF xd#>100 THEN
      PRINT "File will be written without attempting to
        correct for drift"
         GOTO writeout
      END IF
      PRINT "DO YOU WANT TO USE THE TEMPERATURE
CORRECTION
        OF CALIBRATION POINTS (Y/N) ?"
```

```
      PRINT "you may want to run this program again
after
        studying the ";
      PRINT LEFT$(outputfile$,LEN(outputfile$)-4);".DRF
        file"
      DO
         K$=UCASE$(INKEY$)
      LOOP UNTIL K$="Y" OR K$="N"
      IF k$="N" THEN writeout
   END IF

'GOTO writeout ***** use when temperature drift file did
  not work
CLS
LOCATE 5,1,0
?"Adjusting Measured Values"  '-----------------------

'NOTE: we will assume 32767 is at center of range, to
1st
  order so we can use offset & gain (of & gn)
  independently
'
' First calculate offset & gain for each set of
readings,
' then take proportionate part of the ones for beginning
  & end of the Y Axis (j%) table movement;
'  --assume drift evenly distributed over time intervals
    between readings, remember that profile measurements
    take time too.
' Xnew = ( Xold - 32767 ) / Xgn  + 32767 - Xof
' (how nice if numbers were centered on Zero!)

FOR i% = clo% TO chi%
' need the range of corrections from beginning & end of
    Column xof#=x1#(i%)                          'start
   xofd#=(x1#(i%+1)-x1#(i%))/(chi%-clo%+2)     'delta
   vof#=v1#(i%)                                 'start
```

```
   yofd#=(y1#(i%+1)-y1#(i%))/(chi%-clo%+2)     'delta
   xgn#=x2#(i%)                                'start
   xgnd#=(x2#(i%+1)-x2#(i%))/(chi%-clo%+2)     'delta
   ygn#=y2#(i%)                                'start
   ygnd#=(y2#(i%+1)-y2#(i%))/(chi%-clo%+2)     'delta

   FOR j% = clo% TO chi%

x3#(i%,j%)=(x3#(i%,j%)-32767)/(xgn#+xgnd#*(j%-clo%+1))_
                    +32767 - xof#-xofd#*(j%-clo%+1)

y3#(i%,j%)=(y3#(i%,j%)-32767)/(ygn#+ygnd#*(j%-clo%+1))_
                    +32767 - yof#-yofd#*(j%-clo%+1)
   NEXT j%
NEXT I%

'CLS
'LOCATE 5,1,0
'?"Adjusting Extrapolated Values"  '--------------------
'do we really need to ?  - yes for 12" pot at same
  level . . .
'but should be calibrating 12" by then: SO - DON'T
'
CLS
LOCATE 5,1,0
?"Writing Corrected File"  '-----------------------

writeout:

OPEN outputFile$ FOR OUTPUT AS #2

PRINT #2,"!SLA-1 ";nm$;", DX serial # ";dx$;" on ";dt$

PRINT #2,"!Position values used for calibration, & RMS
  deviations (bits)"
PRINT #2,"!Sensor1 ";
```

```
PRINT #2, USING
"######.##";x1tot#,y1tot#,SQR(x1dev#),SQR(y1dev#)
PRINT #2,"!Sensor2 ";
PRINT #2, USING
"######.##";x2tot#,y2tot#,SQR(x2dev#),SQR(y2dev#)

WRITE #2,thi%+1,thi%+1,2,inc#

FOR i% = tlo% TO thi%
   FOR j% = tlo% TO thi%
      A$ = STR$(INT(x3#(i%,j%)*10+0.5)/10)
      PRINT #2,MID$(A$,2);
      PRINT #2,",";
      A$ = STR$(INT(y3#(i%,j%)*10+0.5)/10)
      PRINT #2,MID$(A$,2)
   NEXT j%
NEXT i%
CLOSE #2

CLS
?
?"Calibration correction complete
?
?"We have created two files, ";driftfile$;" and
  ";outputfile$
?
?driftfile$;" has the profiler positions measured after
  each column of the grid"
?
?outputfile$;" has the calibration data which will
become
  STEREO.GEO"
?
?"Rename it C:\3DSYS\STEREO.GEO when loading in
calibrated
  machine"
END
```

```
OutNoSp:
  A$ = STR$(INT(A*10+0.5)/10)
  PRINT #2,MID$(A$,2);
  RETURN^Z
```

```
{=========================================================
CLIFF.PAS
    CLIFF, Control Laser Instrument From File
    for 3D Systems' Sterolithography System

    Version 2.80  by Tarnz Technologies
=========================================================
```

CLIFF reads simple (X,Y) laser position commands from a file and controls the laser mirrors using the LASER background process. Please see the MRCLIFF program for details on the commands and their arguments.

The command line for cliff takes the form:

CLIFF <datafile> [/options]

where <datafile> is the name of the data file containing the laser control information, and /options are any number of user-selectable options:

```
/S    -  show laser control information
/W    -  wait for laser idle at end
/T    -  transparency mode (no CLIFF text)
```

CLIFF can perform the wait-for-idle function (/W option) without having any data file or commands to execute. CLIFF /W and CLIFF /W/T (transparent) will make CLIFF wait for the background task to idle.

<dataf.le> can be replaced by a sequence of direct commands by preceding the commands with a ', ", or @ character. A semicolon can be used to separate commands. As many commands as will fit on one line can be specified. In the following example, the block mode is turned on and the laser beam moved:

```
  CLIFF 'MD BL +; JX 1000; JY 2000; NX 4000; NY 5000; EC

  If CLIFF is called up with no arguments, it will
prompt
    with

  Enter file name: _

  at which point the operator should enter the name of
the data file containing  CLIFF-compatible commands.
Direct commands can be entered here if preceded by a
special character as described above.


============================================================
}

{

  Recent History:
  8/18/87  Ver 2.02  CLIFF Version 2.00 released
    interfaces with memory resident STEREO device driver
  8/22/87  Ver 2.03  /NVC changed to /NC for no version
    check
  8/25/87  Ver 2.04  only updated version number
  9/23/87  Ver 2.12  added MinSystemVersion check
  9/30/87  Ver 2.25  only updated version number
  11/12/87  Ver 2.30  first Beta software release
  12/09/87  Ver 2.40  upgraded to Turbo Pascal Version
4.0
    different STEREO-installed check
  1/25/88  Ver 2.60  version update only
  3/09/88  Ver 2.80  added copyright notice
  4/09/88  Ver 3.00  production release

}
```

```
uses Dos,Crt;

type
  str16 = string[16];
  str80 = string[80];
  str255 = string[255];

var
  ch,key: Char;
  Line,temp,cmd: str255;
  StereoVersion: str16;
  count,i,remline,Er,OpNum,code: Integer;
  Format,LineCount,IOerr: Integer;
  ErrorFile,InputFile: text;
  VersionCheckFlag,TransparentFlag,WaitForIdleFlag:
Boolean;
  ShowData,ErrorOpen,Abort: Boolean;
  regs: Registers;

const
  ThisCliffVersion = '3.00';
  MinSystemVersion = '2.40';
  Ctrl C = #3;
  sp40 = '                                        ';
  _LASERVECTOR  = $64;
  _STEREOVECTOR = $65;

label ExitProgram;

function upper(text: str255): str255;
  var
    i: Integer;
    temp: str255;
  begin
    temp := text;
    for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
```

```
    upper := temp;
  end;

function many(Ch: Char;num: Integer): str255;
  var
    temp: str255;
    i: Integer;
  begin
    temp := '';
    for i:=1 to num do temp  := temp+Ch;
    many := temp;
  end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

procedure print(text: str255);
  begin
    if not TransparentFlag then write(text);
  end;

procedure println(text: str255);
  begin
    if not TransparentFlag then writeln(text);
  end;

function IntStr(num: Integer): str16;
  var
    temp: str16;
  begin
```

```
    if num>=0 then Str(num,temp) else
Str(num+$8000,temp);
    IntStr := temp;
  end;

function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    __LaserOk := (CopyStr = 'Tarnz Technologies');
  end;

function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;

function __StereoOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
```

```
  begin
    with regs do begin
      AX := ($35 shl 8) or _STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      __StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;

function __StereoPresent: Boolean;
  begin
    if __StereoOk then __StereoPresent := true
    else begin
      __StereoPresent := false;
      writeln('STEREO driver not installed.');
    end;
  end;

procedure Error(text: str255);
  begin
    if whereX > 1 then println('');
    if LineCount=0 then println('*** '+text)
      else println('*** '+text+' in line
'+IntStr(LineCount));
    if not ErrorOpen then begin
      rewrite(ErrorFile);
```

```
      ErrorOpen := true;
    end;
    writeln(ErrorFile,text);
  end;

procedure DescribeError(Er: Integer);
  begin
    case Er of
        0: ;
        1: Error('1. Laser interface card not found');
        2: Error('2. Laser channel not open');
        3: Error('3. Background Command Queue full
           timeout');
        4: Error('4. Invalid interrupt time');
        5: Error('5. Not in block mode');
        6: Error('6. Exec Queue limit timeout');
      127: Error('127. Bad SIH command parameter');
      128: Error('128. Unrecognized STEREO command');
      255: Error('255. Internal error (please report)');
      else Error(IntStr(Er)+'. Unrecognized SIH error');
    end;
  end;

{ - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    Pass Command on to Memory Resident Portion

- - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure StereoCommand(line: str255);
  begin
    if (length(line)>0) and (line[1]<>'!') then with
regs
      do begin
      line := line + chr(0);  { end-of-line }
      AX := $00;              { for future expansion }
```

```
      BX := Ofs(line)+1;  { pass pointer to string }
      ES := Seg(line);
      Intr(_STEREOVECTOR,regs);
    end;
    if regs.AL>0 then DescribeError(regs.AL);
  end;

procedure ProcessFile;
  begin
    if not ShowData then begin
      println('');
      print('0    lines');
    end;
    while not eof(InputFile) and not Abort do begin
      readln(InputFile,Line);
      if ShowData then println(Line);
      LineCount := LineCount+1;
      StereoCommand(Line);
      if not TransparentFlag then
        if not ShowData then if LineCount mod 25=0 then
          begin
          gotoxy(1,whereY);
          print(IntStr(LineCount));
        end;
      if keypressed then begin
        key := ReadKey;
        if key in [CtrlC,'A','a'] then Abort := true;
      end;
    end; {not eof}
    if not (TransparentFlag and ShowData) then begin
      gotoxy(1,whereY);
      println(IntStr(LineCount));
    end;
    if Abort then begin
      Error('Control Process Aborted');
      while keypressed do key := ReadKey;
    end; {if Abort}
```

```
  end;

procedure CheckForOptions(var str: str255);
  var
    i: Integer;
  begin
    while (length(str)>1) and (str[1]=' ') do
delete(str,1,1);
    i := pos('/S',upper(str));  { show input file lines
}
    if i>0 then begin
      ShowData := true;
      delete(str,i,2);
    end;
    i := pos('/W',upper(str));  { set wait for idle
      flag }
    if i>0 then begin
      WaitForIdleFlag := true;
      delete(str,i,2);
    end;
    i := pos('/T',upper(str));  { set transparency flag
}
    if i>0 then begin
      TransparentFlag := true;
      delete(str,i,2);
    end;
    i := pos('/NC',upper(str));  { no version check }
    if i>0 then begin
      VersionCheckFlag := false;
      delete(str,i,3);
    end;
  end;

procedure CliffHeader;
  begin
    writeln;
    writeln(#213,many(#205,53),#184);
```

```
    writeln(#179,'  CLIFF, Control Laser Instrument From
      File Program  ',#179);
    writeln(#179,'  Version ',ThisCliffVersion+'
      ',#179);
    writeln(#179,'  Copyright (C) 1988 by Tarnz
      Technologies ',#179);
    writeln(#179,'  3D Systems Laser Sterolithography
      System    ',#179);
    writeln(#212,many(#205,53),#190);
    writeln;
  end;

begin
  ReadCommandLine(cmd);
  LineCount        := 0;
  Abort            := false;
  WaitForIdleFlag  := false;
  TransparentFlag  := false;
  ErrorOpen        := false;
  ShowData         := false;
  VersionCheckFlag := true;

{ ***** check for memory resident STEREO device driver }

  Assign(ErrorFile,'CLIFF.ERR');
  {$I-}  { erase error file if it exists }
  Erase(ErrorFile);
  {$I+}
  IOerr := IOresult;  { clear I/O result }
  CheckForOptions(cmd);

  if not TransparentFlag then CliffHeader;
  if not (__LaserPresent and __StereoPresent) then goto
    ExitProgram;
  if VersionCheckFlag then
    if StereoVersion < MinSystemVersion then begin
      writeln('Obsolete STEREO in use.');
```

```
      goto ExitProgram;
    end;
  if length(cmd)=0 then begin
    write('Enter file name: ');
    readln(cmd);
    CheckForOptions(cmd);
  end;
  if (length(cmd)>1) and (cmd[1] in ['@','"',''''']) then
    begin
    { handle direct commands }
    cmd := copy(cmd,2,199);
    StereoCommand(cmd);
  end else begin
    { handle commands from a file }
    if length(cmd)>0 then begin
      if pos('.',cmd)=0 then cmd := cmd + '.DAT';
      assign(InputFile,cmd);
      {$I-}
      reset(InputFile);
      IOerr := IOresult;
      {$I+}
      if IOerr <> 0 then
        Error('I/O Error '+IntStr(IOerr)+' opening file
          '+cmd)
      else begin
        ProcessFile;
        close(InputFile);
      end;
    end;
  end;
  if WaitForIdleFlag then StereoCommand('WI 1000');

ExitProgram:
  {$I-}
  close(InputFile);
  close(ErrorFile);
  {$I+}
```

```
end.
^Z
```

```
LASER.ASM
        Page 60,132
        Title 3D Systems Laser Driver Package
;=====================================================
;
;
; LASER    3D Systems Laser Controller Driver Package
;          Version 3.00
;
; This code constitutes levels 1 and 2, the lowest
; software levels, of the Laser Controller System.
;
; Copyright (C) Tarnz Technologies, 1987
;
;
;=====================================================

;
;  History:
;
; 8/17/87  Ver 2.02  LASER released
;   new identifcation schemes
; 8/22/87  Ver 2.03  version number updated only
; 8/25/87  Ver 2.04  version number updated only
; 11/12/87  Ver 2.30  first Beta software release,
version
    update only
; 1/25/88  Ver 2.60  version update only
; 4/13/88  Ver 3.00  version update only
;

;
; System Equates
;
current_version equ     0200h    ;current software
version
debug           equ     0        ;debug flag
```

```
;=====================================================
;
; Public And External Declarations
;

;
; public code references
;
public  laser_entry  ;only callable entry point to
                       this pkg
public  enddata ;segment ends for residency size calc
public  endqueue
public  endbkgtask
public  endcode

;
;  public variable references
;
public  laser_int,old_tv,old_tv,current_status
public  sys_int_count,sint_counter,bkgnd_intr_time

;
;  external references
;
extrn   bkg_ins:far ;install background task
extrn   bkg_ins1:far  ;change timebase
extrn   bkg_rmv:far ;remove background task

subttl  Operating System Equates
page
;=====================================================
;
; IBM PC and Compatible Operating System Equates
;
```

```
;
; Equates defining 8259 (Programmable Interrupt
    Controller) and 8253
; (Programmable Interval Timer) ports and constants.
;
inta00          equ     20h         ;8259 PORT
inta01          equ     21h         ;YET ANOTHER 8259 PORT
tim_cmd_reg     equ     43h         ;8253 command register
tim_ch2         equ     42h         ;8253 channel two
ppi_port_A      equ     60h         ;8255 port A
ppi_port_B      equ     61h         ;8255 port B
ext_bkg         equ     4           ;ext background intr
                                    signal
ppi_port_C      equ     62h         ;8255 port C

subttl  Laser I/O Card Equates
page
;=====================================================
;
; Laser I/O Card Equates
;

;
; using John Bell Engineering Card
; "Universal I/O"  83-064
;
laserIO         equ     300h
portA1          equ     laserIO     ;DAC value low byte
portB1          equ     laserIO+1   ;DAC value high byte
portC1          equ     laserIO+2
; outputs
strobeX         =       1           ;X-axis DAC strobe
                                    (falling edge)
strobeY         =       2           ;Y-axis DAC strobe
                                    (falling edge)
strobeZ         =       4           ;Z-axis DAC strobe
                                    (falling edge)
```

```
shutter_ctrl    =       8           ;laser shutter
control
                                    (open/closed)
; inputs
posackXY        =       16          ;X and Y-axis
position
                                     acknowledge
posackZ         =       32          ;Z-axis position
                                    acknowledge
ctrl1           equ     laserIO+3   ;control #1 byte
(write
                                    only!)
portA2          equ     laserIO+4
portB2          equ     laserIO+5
portC2          equ     laserIO+6
ctrl2           equ     laserIO+7
portA3          equ     laserIO+8
portB3          equ     laserIO+9
portC3          equ     laserIO+10
ctrl3           equ     laserIO+11

;
; Miscellaneous
;
ExtSigs         segment at 0F600h    ;for debugging
IntBeg          db      ?
IntEnd          db      ?
ExtSigs         ends

subttl  Miscelaneous Other Equates And Structures
page
;=====================================================
;
; Error codes
;
err_none        equ     0           ;no error
```

```
err_no_card     equ     1           ;no laser interface
card found
err_not_open    equ     2           ;laser channel not
open
err_queue_full  equ     3           ;background command
queue is full
err_bad_time    equ     4           ;invalid interrupt
time
err_not_block   equ     5           ;not in block mode
err_exec_limit  equ     6           ;exec limit (# EC &
                                    EX's) reached
err_bad_param   equ     127         ;some other bad
                                    parameter
err_bad_code    equ     255         ;internal code error

all             equ     0FFFFh      ;for turning flags
off
                                    (all-flag)

subttl  Structures
page
;=====================================================
;
; The stack frame defines the locations of the
parameters
    on the stack
;
stack_frame     struc
p_bp            dw      0
p_dx            dw      0
p_cx            dw      0
p_bx            dw      0
p_ax            dw      0
p_ds            dw      0
p_si            dw      0
p_di            dw      0
p_es            dw      0
```

```
p_flags         dw      0
p_ip            dw      0
p_cs            dw      0
stack_frame     ends

subttl  LASER code segment
page

if1
%out    - Starting pass 1 -
endif
if2
%out    - Starting pass 2 -
endif

subttl  Data Segments
page
LaserData segment public 'DATA'

;
;  Block Variables
;
;  Execute Queue (must be first item in segment)
;  - 10 entries of StartPtr [2 bytes], EndPtr [2 bytes]
;
exec_queue_size         =       200     ;max # of blocks
                                        awaiting
processing
exec_entry_len          =       4
exec_queue_limit        =
exec_queue_size*exec_entry_len
exec_queue              db      exec_queue_limit dup (?)
                                ;queue space

exec_queue_start        dw      0
exec_queue_end          dw      0
```

```
;
;  Background Command Queue Variables
;
bcmd_queue_start        dw      0       ;command queue's
                                        "hard" limits
bcmd_queue_end          dw      0       ;real start and
                                        end of queue

;
;  Execution Block Variables
;
queue_start             dw      0       ;command queue's
                                        "soft" limits
queue_end               dw      0       ;start and end
                                        of current block

rem_queue_start         dw      0       ;redraw storage
of
                                        the queue_start
rem_queue_end           dw      0       ;and queue_end
                                        pointers

;
;  Background Task Variables
;
_static         =       0               ;in static state
_off_delay      =       2               ;in laser
turn-off
                                        state
_idle           =       4               ;in laser system
                                        fully idle state
_pos_ack        =       6               ;in position
                                        acknowledge
state
_on_delay       =       8               ;in laser
turn-on
                                        state
```

```
_moving          =       10              ;in moving state
_pausing         =       12              ;in pausing
state
_redraw_pausing  =       14              ;redrawing pause
                                         state
bkgnd_mode       dw      _static         ;execution mode
bkgnd_intr_time  dw      1190            ;interrupt
                                         interval time
sys_int_count    dw      10              ;sys int routine
                                         tick count
sint_counter     dw      10              ;sys int
                                         down-counter
clear_pointer    dw      0               ;start of
command
                                         queue block
delay            dw      0               ;general delay
                                         downcounter
off_delay_count  dw      0               ;laser turn-off
                                         delay count
value
on_delay_count   dw      0               ;laser turn-on
                                         delay count
value
HomeXpos         dw      0               ;laser beam
(X,Y)
                                         home position
HomeYpos         dw      0
RemXpos          dw      0               ;remember beam
                                        (X,Y) position
here
RemYpos          dw      0

XposFract        db      0               ;current
vector's
                                         variables
Xpos             dw      0
YposFract        db      0
```

```
Ypos            dw      0
deltaX          dw      0
deltaY          dw      0
steps           dw      0
signX           db      0               ;sign-extends
signY           db      0
dummy           dw      0
ut_steps        dw      0               ;utility
                                        downcounting
steps

redraw_count    dw      1               ;redraw pass
count
redraw_delay    dw      0               ;redraw delay
step_periods    dw      10 dup (1190)   ;multi-pass Step
                                        Periods array
mark_position   dw      0               ;redraw mark
                                        position
pass            dw      0               ;redraw pass
                                        number

;
;  Software Interrupt Variables
;
swi_int         dw      0               ;interrupt
vector
                                        number
swi_ofs         dw      0               ;old SWI vector
swi_seg         dw      0

return_code     db      0               ;function return
                                        code

ls_open         =       1               ;laser system
                                        initialized
```

```
ls_interrupt    =       2               ;software
                                        interrupt vector
                                        set
bkgnd_idle      =       4               ;background in
                                        idle state flag
redrawing       =       8               ;doing redraw
                                        operation
need_to_pause   =       16              ;need to pause
                                        flag (redraw)
current_status  dw      0               ;general status
                                        register
autoshut_mode   =       1               ;automatically
                                        open/close
shutter
                                        on idle
posack_mode     =       2               ;wait for X-Y
                                        mirrors position
                                        acknowledge
block_mode      =       4               ;execution block
                                        mode (EX/EC/CL
                                        active)
shutter         =       16              ;open/closed
                                        shutter flag
laser_control   dw      shutter         ;laser control
                                        word

enddata label far
LaserData ends

LaserQueue segment public 'DATA'
;
;  Background Command Queue
;  60K workspace can hold approximately 3750 move
entries
;
bcmd_queue_size         =       60000   ;vector circular
                                        queue size
```

```
bcmd_queue_limit        =       bcmd_queue_size-32
bcmd_queue              db      bcmd_queue_size+16 dup
(?)
move_size               =       13      ;background
                                        command sizes
open_shutter_size       =       1
close_shutter_size      =       1
change_timebase_size    =       3
pause_size              =       3
set_periods_size        =       2       ; + 2 * num
set_redraw_count_size   =       3
set_redraw_delay_size   =       3
mark_size               =       1
redraw_size             =       1

endqueue label far
LaserQueue ends

subttl  Laser Background Task
page
;=====================================================
;
; Laser Control Background Task
;
; responsible for:
;
; - updating x-axis & y-axis DACs every 100 us (typ.)
; - moving of laser beam
; - geometric correction
; - shutter  ontrol
; - idling s  tem while command queue is empty
;
;=====================================================
LaserGroup group LaserBackground,LaserCode

LaserBackground segment public 'CODE'
  assume  cs:LaserBackground,ds:LaserData
```

```
;
; high-speed timer 0 interrupt
;
; typically 100 us, settable via software interrupt
;
laser_int proc far
  assume  cs:LaserBackground,ds:LaserData,es:LaserData

  ;
  ;  set for interrupt service
  ;
  push    ax
; in      al,ppi_port_B           ;signal start of
background interrupt
; and     al,all-ext_bkg
; out     ppi_port_B,al
  push    ds
  push    bx
  push    cx
  push    dx
  push    es
  push    di
  push    si

  ;
  ;  background task interrupt service
  ;
  mov     ax,LaserData            ;point to laser system
                                  variables
  mov     ds,ax
  mov     ax,LaserQueue           ;point to laser
command
                                  queue
  mov     es,ax
  cld                             ;for block moves
  mov     bx,bkgnd_mode
  jmp     word ptr cs:[bx + offset bkgnd_states]
```

```
  ;
  ;  Background State Address Table
  ;'
  bkgnd_states label word
  dw      static
  dw      off_delay
  dw      idle
  dw      pos_ack
  dw      on_delay
  dw      moving
  dw      pausing
  dw      redraw_pausing

  ;
  ;  old system timer interrupt vector
  ;  must be in CS for indirect intersegment jump to
BIOS
  ;
old_tv  dd      ?                       ;called every
sys_int_count ticks

  ;
  ;  static state - determine processing
  ;
static:
   test    laser_control,block_mode
   jnz     block

;
;  STREAM MODE
;  laser commands executed as they are added to table
for
     best throughput
;  and easiest control
;

stream:
```

```
  mov     si,bcmd_queue_start     ;any entries in queue?
  cmp     si,bcmd_queue_end
  jz      prep_idle
  jmp     read_queue              ;yes, go execute
command

;
; BLOCK MODE
; laser commands executed in blocks to prevent
    intermediate idles
; multi-pass redraw of small sets of vectors supported

block:
  mov     ax,queue_start          ;finished processing
                                  block?
  cmp     ax,queue_end
  jz      block1
  jmp     read_queue
block1:
  test    current_status,redrawing        ;redrawing?
  jnz     go_redraw
  mov     si,exec_queue_start     ;no, something in exec
                                   queue?
  cmp     si,exec_queue_end
  jz      prep_idle               ;no, idle
  jmp     block_proc              ;yes, go process next
                                  block

go_redraw:
  jmp     redraw_pass             ;multiple passes

  ;
  ;  prepare to idle
  ;
prep_idle:
  test    laser_control,autoshut_mode  ;if autoshutter
          mode, close shutter
```

```
  jz      home
  or      laser_control,shutter
  call    set_ctrl_port
  jmp     laser_off
home:
  mov     ax,Xpos       ;  else remember current position
  mov     RemXpos,ax
  mov     ax,Ypos
  mov     RemYpos,ax
  mov     ax,HomeXpos              ;     and home laser
beam
  mov     Xpos,ax
  mov     ax,HomeYpos
  mov     Ypos,ax
  call    update_DACs

laser_off:
  Mov     ax,off_delay_count ;observe laser turn-off
delay
  cmp     ax,0
  je      set_idle             ;if 0, don't delay at all
  mov     delay,ax
  mov     bkgnd_mode,_off_delay
  sdone:  jmp     int_fini

  ;
  ;  laser turn-off delay state
  ;
off_delay:
  dec     delay
  jnz     sdone

set_idle:
  mov     bkgnd_mode,_idle                ;set idle
state

  ;
```

```
  ;  laser system fully idle state
  ;
idle:
  test    laser_control,block_mode
  jz      idle_stream

idle_block:
  mov     si,exec_queue_start     ;wait for something in
                                   exec queue
  cmp     si,exec_queue_end
  jnz     out_of_idle

idle_set:
  or      current_status,bkgnd_idle   ;make sure idle
flag
                                      is on
  jmp     int_fini

idle_stream:
  mov     si,bcmd_queue_start     ;wait for something in
                                  command queue
  cmp     si,bcmd_queue_end
  jz      idle_set

out_of_idle:
  and     current_status,all-bkgnd_idle   ;turn off idle
                                           flag
  test    laser_control,autoshut_mode ;if autoshut mode,
                                        open shutter
  jz      repos
  and     laser_control,all-shutter
  call    set_ctrl_port
  jmp     laser_on
repos:
  mov     ax,RemXpos      ;   else reposition laser beam
  mov     Xpos,ax
  mov     ax,RemYpos
```

```
  mov     Ypos,ax
  call    update_DACs

  test    laser_control,posack_mode
  jz      laser_on
  mov     dx,portC1     ;wait for position
acknowledgement
  in      al,dx
  and     dx,posackXY
  jz      laser_on
  mov     bkgnd_mode,_pos_ack

  ;
  ;  position acknowledge state
  ;
pos_ack:
  mov     dx,portC1
  in      al,dx
  and     dx,posackXY
  jnz     sdone
  mov     bkgnd_mode,_static

laser_on:
  mov     ax,on_delay_count  ;observe laser turn-on
delay
  cmp     ax,0
  je      set_proc                ;if 0, don't delay
  mov     delay,ax
  mov     bkgnd_mode,_on_delay
  jmp     int_fini

  ;
  ;  laser turn-on delay state
  ;
on_delay:
  dec     delay
  jz      set_proc
```

```
  jmp     int_fini
set_proc:
  mov     bkgnd_mode,_static      ;set for normal
                                   processing
  test    laser_control,block_mode
  jz      read_queue              ;start processing
commands immediately

  ;
  ;  setup for block mode processing
  ;
block_proc:
  mov     si,exec_queue_start
  mov     ax,[si]                 ;get start index
  mov     bcmd_queue_start,ax     ;update command
queue's
                                  hard start
  mov     queue_start,ax          ;set exec block start
  add     si,2
  mov     ax,[si]                 ;get end index
  mov     queue_end,ax            ;set exec block end
  add     si,2
  cmp     si,exec_queue_limit     ;next index
  jb      block_proc1
  mov     si,0
block_proc1:
  mov     exec_queue_start,si
  jmp     block                  ;catch 0-length blocks

  ;
  ;  begin processing background command from command
     queue
  ;
read_queue:
  test    laser_control,block_mode
  jz      read_stream
read_block:
```

```
  mov     si,queue_start          ;use right pointer
  jmp     read_queue1
read_stream:
  mov     si,bcmd_queue_start
read_queue1:
  mov     al,es:[si]              ;read next background
                                  command
  inc     si                      ;point to command
parameters
  inc     al                      ; + 1 so that FF becomes 0
  xor     ah,ah                   ;compute jump tableoffset
  shl     ax,1
  cmp     ax,bkgnd_ops_end        ;don't use to increase
                                    throughput
  jae     skip_bad_token
  mov     bx,ax
  jmp     word ptr cs:[bx + offset bkgnd_ops_tbl]

skip_bad_token:
  mov     bx,1                    ;skip past bad token
  jmp     end_read_queue          ;  hope we resync on
next byte

bkgnd_ops_tbl label word
  dw      reset_queue             ;FF
  dw      move                    ;0
  dw      open_shutter            ;1
  dw      close_shutter           ;2
  dw      change_timebase         ;3
  dw      pause                   ;4
  dw      set_periods             ;5
  dw      set_redraw_count        ;6
  dw      set_redraw_delay        ;7
  dw      mark                    ;8
  dw      redraw                  ;9
bkgnd_ops_end equ $-bkgnd_ops_tbl
```

```
;  ---------------------------------------------
;
; reset to beginning of command queue
; Background Command FF (Internal)
;
;  ---------------------------------------------
reset_queue:
  mov     queue_start,0           ;zero start pointer
  jmp     static                  ;go process next
command, if any

;  ---------------------------------------------
;
; move/draw vector
; Background Command 0
;
;  ---------------------------------------------
move:
  mov     bx,ds                   ;swap DS & ES
                                  temporarily for movsw
  mov     ax,es
  mov     ds,ax
  mov     es,bx
  mov     di,offset XposFract
  mov     cx,(move_size+1)/2      ;copy packet from
queue
                                   to workarea
 rep movsw
  mov     bx,ds                   ;swap segments back
  mov     ax,es
  mov     ds,ax
  mov     es,bx
  mov     al,byte ptr deltaX+1    ;compute sign-extends
  cbw
  mov     signX,ah
  mov     al,byte ptr deltaY+1
  cbw
```

```
  mov     signY,ah
  test    current_status,need_to_pause
  jz      move1
  jmp     redraw_pause
move1:
  mov     bkgnd_mode,_moving
  mov     bx,move_size

;
;  end of reading queue
;
end_read_queue:
  test    laser_control,block_mode
  jz      end_stream

end_block:
  mov     ax,queue_start
  add     ax,bx
  cmp     ax,bcmd_queue_size      ;next start index
  jb      end_read_queue1
  mov     ax,0                    ;wraparound command
queue (soft)
end_read_queue1:
  mov     queue_start,ax
  jmp     set_DACs

end_stream:
  mov     ax,bcmd_queue_start
  add     ax,bx
  cmp     ax,bcmd_queue_size      ;next start index
  jb      end_read_queue2
  mov     ax,0                    ;wraparound command
queue (hard)
end_read_queue2:
  mov     bcmd_queue_start,ax
  jmp     set_DACs
```

```
  ;
  ;  moving state - vector processing
  ;
moving:
  mov     ax,word ptr XposFract   ;compute new X
position
  add     ax,deltaX               ;add fract & 1st byte
of
                                  integer
  mov     word ptr XposFract,ax
  mov     al,byte ptr Xpos+1      ;add carry to 2nd byte
                                   of integer
  adc     al,signX
  mov     byte ptr Xpos+1,al

  mov     ax,word ptr YposFract   ;compute new Y
position
  add     ax,deltaY
  mov     word ptr YposFract,ax
  mov     al,byte ptr Ypos+1
  adc     al,signY    ;0 for + deltas, FFh for - deltas
  mov     byte ptr Ypos+1,al

  dec     steps    ;count down # steps
  jnz     set_DACs
  mov     bkgnd_mode,_static      ;static state next
tick

  ;
  ;  set laser mirror DACs to appropriate values
  ;
set_DACs:
  call    update_DACs

  ;
  ;  return from interrupt
  ;
```

```
int_fini:
  dec     sint_counter      ;call system timer
                            interrupt routine
  mov     ax,sint_counter   ; every (sys_int_count)
                            # ticks
  cmp     ax,0
  je      sys_fini
  pop     si   ;normal, non-sys interrupt return
  pop     di
  pop     es
  pop     dx
  pop     cx
  pop     bx
  pop     ds
  mov     al,20h                ;acknowledge interrupt
  out     020h,al
  sti
  pop     ax
  iret

sys_fini:
  mov     ax,sys_int_count      ;reload downcounter
  mov     sint_counter,ax
  pop     si
  pop     di
  pop     es
  pop     dx
  pop     cx
  pop     bx
  pop     ds
  mov     al, 0h                ;reset interrupt to
                                allow laser
  out     020h,al               ;interrupts during
BIOS
                                operation
  sti
  pop     ax
```

```
  jmp     old_tv     ;continue timer interrupt in BIOS

bkgnd_local_sbrs proc near
page
;
;  Set Output Control Port
;    - shutter control
;
set_ctrl_port:
  mov     dx,portC1
  in      al,dx      ;read existing port C1 bits
  test    laser_control,shutter   ;update for shutter
  jnz     shut_closed
shut_open:
  and     al,all-shutter_ctrl
  jmp     shut_str
shut_closed:                              ;shutter closed
if shutter not 0
  or      al,shutter_ctrl
shut_str:
  out     dx,al
  ret

;
;  Update laser beam X and Y Digital to Analog
Converters
;
update_DACs:

set_XDAC:
  mov     ax,Xpos        ;output X-pos to laser X-axis
DAC
  mov     dx,portA1
  out     dx,al
  inc     dx
  mov     al,ah
  out     dx,al
```

```
        inc     dx                      ;port C1 has strobe
      lines
        in      al,dx
        and     al,all-strobeX          ;falling edge
        out     dx,al
        or      al,strobeX
        out     dx,al

      set_YDAC:
        mov     ax,Ypos                  ;output Y-pos to laser
      Y-axis DAC
        mov     dx,portA1
        out     dx,al
        inc     dx
        mov     al,ah
        out     dx,al
        inc     dx                      ;port C1 has strobe
      lines
        in      al,dx
        and     al,all-strobeY          ;falling edge
        out     dx,al
        or      al,strobeY
        out     dx,al

        mov     ax,0FFh
        mov     dx,portA1
        out     dx,al
        inc     dx
        out     dx,al
        ret

      bkgnd_local_sbrs endp
      page
      ;  ---------------------------------------------
      ;
      ; open laser beam shutter
      ; Background Command 1
```

```
;
;  ---------------------------------------------
open_shutter:
  and     laser_control,all-shutter
  call    far ptr set_ctrl
  mov     bx,open_shutter_size
  jmp     end_read_queue

;  ---------------------------------------------
;
; close laser beam shutter
; Background Command 2
;
;  ---------------------------------------------
close_shutter:
  or      laser_control,shutter
  call    far ptr set_ctrl
  mov     bx,close_shutter_size
  jmp     end_read_queue
;  ---------------------------------------------
;
; change interrupt time base
; Background Command 3
;
;  ---------------------------------------------
change_timebase:
  mov     bx,es:[si]
  mov     step_periods,bx   ;just alter first step
period
  call    bkg_ins1                ;change time base now
  mov     bx,change_timebase_size
  jmp     end_read_queue

;  ---------------------------------------------
;
; pause queue processing
; Background Command 4
```

```
;
;  ---------------------------------------------
pause:
  mov     ax,es:[si]
  mov     ut_steps,ax
  mov     bkgnd_mode,_pausing     ;set for pausing state
  mov     bx,pause_size
  jmp     end_read_queue

pausing:
  dec     ut_steps                ;count down # steps
  jnz     pausing1
  mov     bkgnd_mode,_static      ;static state next
tick
pausing1:
  jmp     int_fini

;  ---------------------------------------------
;
; set_periods
; Background Command 5
;
;  ---------------------------------------------
set_periods:
  mov     cl,es:[si]              ;get sp count [1 byte]
  inc     si
  mov     ch,0                    ;set high byte to 0
  push    cx
  cmp     pass,0                  ;read sp only on
redraw    jnz     set_periods_skip
  mov     di,offset step_periods
  mov     bx,es:[si]              ;get sp0
  mov     [di],bx
  push    es
  call    bkg_ins1                ;set new timer
interrupt
  pop     es
```

```
  jmp     set_periods_next
set_periods_loop:
  mov     ax,es:[si]              ;copy sp1, sp2...
  mov     [di],ax                 ;  to step_periods
array
set_periods_next:
  add     si,2
  add     di,2
  dec     cl
  jnz     set_periods_loop
set_periods_skip:
  pop     bx                      ;calc size of sp
command
  shl     bx,1
  add     bx,set_periods_size
  jmp     end_read_queue

;  ---------------------------------------------
;
; set_redraw_count
; Background Command 6
;
;  ---------------------------------------------
set_redraw_count:
  mov     ax,es:[si]
  mov     redraw_count,ax
  mov     bx,set_redraw_count_size
  jmp     end_read_queue

;  ---------------------------------------------
;
; set_redraw_delay
; Background Command 7
;
;  ---------------------------------------------
set_redraw_delay:
  mov     ax,es:[si]
```

```
  mov     redraw_delay,ax
  mov     bx,set_redraw_delay_size
  jmp     end_read_queue

;  ------------------------------------------------
;
;  mark
;  Background Command 8
;
;  ------------------------------------------------
mark:
  mov     ax,queue_start ;remember current queue
position
  mov     mark_position,ax
  mov     bx,mark_size
  jmp     end_read_queue

;  ------------------------------------------------
;
;  redraw
;  Background Command 9
;
;  ------------------------------------------------

  ;
  ;  set up for redraw
  ;
redraw:
  test    laser_control,block_mode
  jz      no_redraw       ;make sure task in block mode
  mov     ax,queue_start
  mov     rem_queue_start,ax  ;remember current queue
           start / end
  mov     ax,queue_end
  mov     rem_queue_end,ax
  or      current_status,redrawing  ;turn on redrawing
           flag
```

```
  ;
  ;  start a redraw pass
  ;
redraw_pass:
  inc     pass                    ;next pass number
  mov     di,pass
  cmp     di,redraw_count         ;all passes done?
  jae     finish_redraw
  shl     di,1                    ; x 2
  mov     bx,step_periods[di]     ;no, change step
period
           (1..)
  call    bkg_ins1
  cmp     redraw_delay,0          ;if redraw_delay > 0,
  je      redraw_pass1
  or      current_status,need_to_pause    ;  pause on
next
           vector pos
redraw_pass1:
  mov     ax,mark_position        ;set redraw start /
end
  mov     queue_start,ax
  mov     ax,rem_queue_start
  mov     queue_end,ax
  mov     bx,mark_size  ;skip mark or last redraw
           instruction
  jmp     end_read_queue

  ;
  ;  finish up redraw
  ;
finish_redraw:
  mov     ax,rem_queue_start   ;restore queue start /
end
  mov     queue_start,ax
  mov     mark_position,ax        ;update mark position
  mov     ax,rem_queue_end
```

```
        mov     queue_end,ax
        and     current_status,all-redrawing-need_to_pause
                  ;turn off flags
        mov     pass,0                  ;set redraw pass to 0
        mov     bx,step_periods         ;set step period 0
        call    bkg_ins1
no_redraw:
        mov     bx,redraw_size  ;skip redraw instruction
        jmp     end_read_queue

redraw_pause:
        mov     ax,redraw_delay ;delay this number of timer
                 ticks
        mov     ut_steps,ax
        mov     bkgnd_mode,_redraw_pausing
        jmp     set_DACs   ;update DACs with first move
position

redraw_pausing:
        dec     ut_steps                ;count down # steps
        jz      redraw_pausing1
        jmp     int_fini
redraw_pausing1:
        and     current_status,all-need_to_pause  ;turn off
                 need-to-pause flag
        jmp     move1                   ;now, go do vector
move

laser_int endp

endbkgtask label far

LaserBackground ends

subttl  Copyright Notice
page
LaserCode segment public 'CODE'
```

```
  assume  cs:LaserCode,ds:LaserData

;=======================================================
;
; LASER SYSTEM INFORMATION AREA
;
  dw      current_version          ;current version
number
  dw      LaserData        ;memory-resident segments
  dw      LaserQueue
  dw      LaserBackground
  dw      LaserCode
  db      16 dup (0)

;=======================================================
;
; COPYRIGHT NOTICE
;
  db  'LASER Controller Version 3.00 Copyright (C) 1988
by
         Tarnz Technologies '

subttl  Software Interrupt Service
page
;=======================================================
;
; Laser Software Interrupt Functions
;
; These functions can be called from any high-level
    language having
; software interrupt capability.
;
; AH AL
; 00 --   open laser DAC channel and initialize
; 01 --   close laser DAC channel and restore system
; 02 xx   add command to background command queue
; 02 00     move/draw vector
```

```
; 02 01     shutter open
; 02 02     shutter closed
; 02 03     change time base
; 02 04     pause
; 03 --     reset background command queue
; 04 --     get background command queue size
; 05 xx     get_option
; 05 00     get software interrupt vector number
; 05 01     get background task interval time
; 05 02     get laser control word
; 05 03     get laser status
; 05 04     get laser turn-off (active to idle) delay
; 05 05     get laser turn-on (idle to active) delay
; 05 06     get laser home position
; 06 xx    set_option
; 06 00     set software interrupt vector number
; 06 01     set background task interval time
; 06 02     set laser control word
; 06 03     -
; 06 04     set laser turn-off delay
; 06 05     set laser turn-on delay
; 06 06     set lase home position
; 07 xx     background task execution control
; 07 00     execute command queue to current point and
               stop
; 07 01     execute command queue to current point,
stop,
               and clear
; 07 02     clear command queue (reset queue's start
               pointer)
; 08 xx     background task redraw control
; 08 00     set multi-pass step periods
; 08 01     set redraw pass count
; 08 02     set redraw delay
; 08 03     mark position
; 08 04     perform multi-pass redraw
;
```

```
;=======================================================
page
;=======================================================
;
; Laser Controller Software Interface
; Interrupt Handler
;
; All calls have the following format:
;
; ah = function code
; al = sub-function code if applicable
;
; bx..dx are call dependent
;
; On return, the following is always true:
;
; al = error code:        0 - operation was successful
;                   255 - internal error, please report
;
;=======================================================

;
; software interrupt entry point
;
laser_swi:
  call    far ptr laser_entry
  iret

laser_entry proc far
  ;
  ;  save the working registers on the user's stack for
  ;  the moment until we can switch to the internal
stack
  ;
  pushf
  push    es
  push    di
```

```
        push    si
        push    ds
        push    ax
        push    bx
        push    cx
        push    dx
        push    bp
        mov     bp,sp

        mov     ax,LaserData  ;point to laser system variables
        mov     ds,ax
        mov     ax,LaserQueue    ;point to laser system queue
        mov     es,ax

        ;
        ;  decode the function call and dispatch it
        ;
        mov     ax,[bp].p_ax
        mov     return_code,err_none

        ;
        ;  convert AH into an index into the function table
        ;
        mov     al,ah          ;compute jump table offset
        xor     ah,ah
        shl     ax,1
        cmp     ax,funct_tbl_end
        jc      entry_1
        mov     return_code,err_bad_param   ;invalid function
                   code
        jmp     entry_2

        ;
        ;  call the appropriate procedure
        ;
      entry_1:
        mov     bx,ax
```

```
    call    word ptr cs:[bx + offset funct_tbl]

    ;
    ;  return to caller with return code
    ;
entry_2:
    mov     ax,[bp].p_ax
    mov     al,return_code
    mov     [bp].p_ax,ax
    mov     sp,bp
    pop     bp
    pop     dx
    pop     cx
    pop     bx
    pop     ax
    pop     ds
    pop     si
    pop     di
    pop     es
    popf
    ret

laser_entry endp

funct_tbl label word
    dw      open_laser_and_init     ;0
    dw      close_laser_and_reset   ;1
    dw      add_to_bcmd_queue       ;2
    dw      reset_bcmd_queue        ;3
    dw      get_bcmd_queue_size     ;4
    dw      get_option              ;5
    dw      set_option              ;6
    dw      bkgnd_exec_ctrl         ;7
    dw      redraw_ctrl             ;8
funct_tbl_end equ $-funct_tbl

subttl  setinterrupt
```

```
page
;=====================================================
;
; Set an interrupt vector.  The register programming is:
;
; BX:AX = CS:IP for the Interrupt Service Routine
; DX    = interrupt type (number)
; Upon return, BX:AX contains the previous interrupt
    vector
;
;=====================================================
setinterrupt proc near
  push    ds
  push    es
  push    cx
  push    dx
  push    bx
  push    ax
  mov     ax,dx                   ;first return old
vector
  mov     ah,35h                  ;it will be in ES:BX
  int     21h
  mov     cx,bx                   ;save BX
  mov     ax,dx                   ;now set the new
vector
  mov     ah,25h
  pop     dx              ;pop AX into DX and so that
  pop     bx              ;  DS:DX is the desired value
  mov     ds,bx
  push    es                      ;save ES and CX
  push    cx
  int     21h
  pop     ax             ;restore ES:CX as BX:AX and
  pop     bx                      ;  return
  pop     dx
  pop     cx
  pop     es
```

```
  pop     ds
  ret

setinterrupt endp

subttl  Software Interrupt Functions
page

soft_fns proc near

;-------------------------------------------------------
;
; AH = 00:  open_laser_and_init
;
; Install background task and initialize variables.
;
; if BX > 0 then use BX to set new background interrupt
    interval
; else use previously defined time interval value
;
;-------------------------------------------------------
open_laser_and_init:
  call    reset_bcmd_queue
  mov     bx,[bp].p_bx
  or      bx,bx
  jnz     open_1
  mov     bx,bkgnd_intr_time
open_1:
  call    set_ports
  call    set_ctrl
  call    set_bkgnd_time
  ret

;
;  set up 8255 ports on John Bell I/O Card
;
;  A1(0-7)  output  mode 0
```

```
;  B1(0-7)  output  mode 0
;  C1(0-3)  output  mode 0
;  C1(4-7)  input   mode 0
;
set_ports:
  mov     dx,ctrl1
  mov     al,10001000b            ;output control1 byte
  out     dx,al                   ;  doing so reset
ports
; mov     dx,portA1               ;init ports
; mov     al,00h
; out     dx,al                 ;port A1, DAC value of 0
; inc     dx
; out     dx,al                   ;port B1
  inc     dx
  mov     dx,portC1
  mov     al,0FFh
  out     dx,al                   ;port C1, strobes high
  ret
page

;-------------------------------------------------
;
; AH = 01:  close_laser_and_reset
;
; Stop background task and clean-up system as much as
    possible.
;
;-------------------------------------------------
close_laser_and_reset:
  call    bkg_rmv
  mov     bkgnd_mode,_static      ;static state
  ret
page

;-------------------------------------------------
;
```

```
; AH = 02:  add command to background task's queue
;
; AL    COMMAND TYPE ADDED
; 00    vector move/draw
; 01    shutter open
; 02    shutter closed
; 03    change time base
; 04    pause
;
;-----------------------------------------------------
addbcmd_tbl label word
  dw      add_vector
  dw      add_shutter_open
  dw      add_shutter_close
  dw      add_change_timebase
  dw      add_pause
addbcmd_tbl_end equ $-addbcmd_tbl

add_to_bcmd_queue:
  test    current_status,ls_open  ;only add vector if
             channel open
  jnz     add_bcmd1
  mov     return_code,err_not_open
  ret
add_bcmd1:
  call    get_size   ;check for near-full background
queue
  cmp     ax,bcmd_queue_limit
  jb      add_bcmd2
  mov     return_code,err_queue_full
  ret
add_bcmd2:
  mov     ax,[bp].p_ax
  mov     di,bcmd_queue_end
  mov     es:[di],al              ;store command type
  inc     di
  xor     ah,ah
```

```
        shl     ax,1
        cmp     ax,addbcmd_tbl_end
        jc      add_bcmd3
        mov     return_code,err_bad_param   ;invalid function
        ret
        ;
        ;  call appropriate function
        ;
add_bcmd3:
        mov     bx,ax
        call    word ptr cs:[bx + offset addbcmd_tbl]
        ret

;
; AX = 0200:  add_vector
;
; Add vector to background command queue.
;
; ES:BX points to vector packet:
; byte   0 - startX fractional  (start coordinates)
; 1,2 - startX integer
;   3 - startY fractional
; 4,5 - startY integer
; 6,7 - deltaX fract/int  (change coordinates)
; 8,9 - deltaY fract/int
; 10,11 - incr  (# of steps)
;
add_vector:
        push    ds
        mov     ds,[bp].p_es            ;ES:BX point to vector
packet
        mov     si,[bp].p_bx
        cld
        mov     cx,(move_size+1)/2
        rep movsw
        pop     ds
        mov     bx,move_size
```

```
  jmp     end_store

;
; AX = 0201:  background shutter open
;
; Open laser beam shutter in-line with background
      sequencing.
;
add_shutter_open:
  mov     bx,open_shutter_size
  jmp     end_store

;
; AX = 0202:  background shutter closed
;
; Close laser beam shutter in-line with background
     sequencing.
;
add_shutter_close:
  mov     bx,close_shutter_size
  jmp     end_store

;
; AX = 0203:  background change time base
;
;  Change interrupt time base in-line with background
     sequencing.
;
add_change_timebase:
  mov     bx,[bp].p_bx
  mov     es:[di],bx
  mov     bx,change_timebase_size
  jmp     end_store

;
; AX = 0204:  temporarily pause background task
;
```

```
; Pause background processing in-line with background
    sequencing.
;
add_pause:
  mov     bx,[bp].p_bx
  mov     es:[di],bx
  mov     bx,pause_size

;
;  finish off storing to background queue
;
end_store:
  mov     ax,bcmd_queue_end       ;next end index
  add     ax,bx
  cmp     ax,bcmd_queue_limit     ;next start index
  jb      end_store1
  mov     di,ax                   ;store queue max token
  mov     es:[di],byte ptr 0FFh
  mov     ax,0                    ;wraparound queue
end_store1:
  mov     bcmd_queue_end,ax
  ret
page

;-------------------------------------------------
;
; AH = 03:  reset_bcmd_queue
;
; Throw away all commands currently in background
command
    queue.
; Current background task operations are suspended.
;
;-------------------------------------------------
reset_bcmd_queue:
  pushf                   ;will stop current operation
  cli
```

```
  mov     ax,exec_queue_end
  mov     exec_queue_start,ax
  mov     ax,bcmd_queue_end     ;force queues to zero
           length
  mov     bcmd_queue_start,ax   ;  and make background
routine idle
  mov     clear_pointer,ax
  mov     ax,queue_end
  mov     queue_start,ax
  and     current_status,all-redrawing-need_to_pause
;no
           redraw
  mov     pass,0
  mov     bkgnd_mode,_static    ;will enter idle state
            popf
  ret
page

;-----------------------------------------------------
;
; AH = 04:  get_bcmd_queue_size
;
; Return the number of bytes currently used by
background
     command
; queue in BX.
;
;-----------------------------------------------------
get_bcmd_queue_size:
  call    get_size
  mov     [bp].p_bx,ax
  ret

get_size:
  pushf
  cli
  mov     ax,bcmd_queue_end       ;try end-start
```

```
  sub     ax,bcmd_queue_start
  jnc     get_size1
  ;
  ;  add in background command queue size for negative
       numbers
  ;    won't be perfectly accurate on queue wraparound
  ;
  add     ax,bcmd_queue_size
get_size1:
  popf
  ret
page

;-----------------------------------------------------
;
; AH = 05:  get_option
;
; Return laser control option:
;
; AL    OPTION RETURNED
; 00    software interrupt vector number
; 01    background task interval time
; 02    laser control word
; 03    laser status
; 04    laser turn-off delay tick count
; 05    laser turn-on delay tick count
; 06    laser beam home position
;
;-----------------------------------------------------
goptn_tbl       label   word
  dw      get_int_vector
  dw      get_bkgnd_time
  dw      get_laser_ctrl
  dw      get_status
  dw      get_off_count
  dw      get_on_count
  dw      get_home_pos
```

```
goptn_tbl_end   equ     $-goptn_tbl

get_option:
  mov     ax,[bp].p_ax
  xor     ah,ah
  shl     ax,1
  cmp     ax,goptn_tbl_end
  jc      go_1
  mov     return_code,err_bad_param   ;invalid function
  ret

go_1:   ;
  ;  call appropriate function
  ;
  mov     bx,ax
  call    word ptr cs:[bx + offset goptn_tbl]
  ret

;
; AX = 0500:  get software interrupt vector number
;
; BL gets interrupt vector number
;
get_int_vector:
  mov     ax,swi_int
  mov     bx,[bp].p_bx
  mov     bl,al
  mov     [bp].p_bx,bx
  ret

;
; AX = 0501:  get background task interrupt interval
time
;
; BX gets timer parameter:
; intr freq = BX / 1.19 MHz  (e.g. if BX = 119, freq is
     100 us)
```

```
;
;
get_bkgnd_time:
  mov     bx,bkgnd_intr_time
  mov     [bp].p_bx,bx
  ret

;
; AX = 0502:  get laser control word
;
get_laser_ctrl:
  mov     bx,laser_control
  mov     [bp].p_bx,bx
  ret

;
; AX = 0503:  get laser status
;
get_status:
  mov     ax,current_status   ;return current_status in
BX
  mov     [bp].p_bx,ax
  mov     dx,portC1     ;return Port C1 inputs in CX
  in      al,dx
  shl     al,1
  shl     al,1
  shl     al,1
  shl     al,1
  mov     ah,0
  mov     [bp].p_cx,ax
  ret

;
; AX = 0504:  get laser turn-off delay count
;
get_off_count:
  mov     bx,off_delay_count      ;return off tick count
```

```
  mov     [bp].p_bx,bx
  ret

;
; AX = 0505:  get laser turn-on delay count
;
get_on_count:
 mov     bx,on_delay_count       ;return on tick count
 mov     [bp].p_bx,bx
 ret

;
; AX = 0506:  get laser home position
;
get_home_pos:
  mov     bx,HomeXpos
  mov     [bp].p_bx,bx            ;home X-position in BX
  mov     cx,HomeYpos
  mov     [bp].p_cx,cx            ;home Y-position in CX
  ret
page

;-----------------------------------------------------
;
; AH = 06:  set_option
;
; Change laser control option:
;
; AL    OPTION CHANGED
; 00    software interrupt vector number
; 01    background task interval time
; 02    laser control word
; 03    -
; 04    laser turn-off delay tick count
; 05    laser turn-on delay tick count
; 06    laser beam home position
;
```

```
;-----------------------------------------------------
soptn_tbl label word
  dw      set_int_vector
  dw      set_bkgnd_time
  dw      set_laser_ctrl
  dw      no_so
  dw      set_off_count
  dw      set_on_count
  dw      set_home_pos
soptn_tbl_end equ $-soptn_tbl

set_option:
  mov     ax,[bp].p_ax
  xor     ah,ah
  shl     ax,1
  cmp     ax,soptn_tbl_end
  jc      so_1
  mov     return_code,err_bad_param   ;invalid function
  ret

so_1:   ;
  ;  call appropriate function
  ;
  mov     bx,ax
  call    word ptr cs:[bx + offset soptn_tbl]
no_so:
  ret

;
; AX = 0600:  set software interrupt vector number
;
; if BL = 0,
; then if vector previously set, then restore old vector
; else do nothing
; else use BL to set software interrupt vector number
;
set_int_vector:
```

```
        mov     bx,[bp].p_bx
        or      bl,bl
        jz      set_int_1
        xor     bh,bh
        mov     swi_int,bx
        mov     dx,bx
        mov     ax,offset laser_swi
        mov     bx,cs
        call    setinterrupt
        or      current_status,ls_interrupt
        mov     swi_seg,bx
        mov     swi_ofs,ax
        ret

set_int_1:
        ;
        ;  restore old interrupt vector
        ;
        test    current_status,ls_interrupt
        jnz     set_int_2
        ret

set_int_2:
        mov     bx,swi_seg
        mov     ax,swi_ofs
        mov     dx,swi_int
        call    setinterrupt
        and     current_status,all-ls_interrupt
        ret


;
; AX = 0601:  set background task interrupt interval
time
; activate background task if not already running
;
; BX has timer parameter:
```

```
; intr freq = BX / 1.19 MHz  (e.g. if BX = 119, freq is
     100 us)
;
;
set_bkgnd_time:
  mov     bx,[bp].p_bx
  mov     step_periods,bx
  call    bkg_ins
  ret

;
; AX = 0602:  set laser control word
;
set_laser_ctrl:
  mov     bx,[bp].p_bx
  mov     laser_control,bx

  ;
  ;  set control bits
  ;
set_ctrl:
  mov     dx,portC1
  in      al,dx       ;read existing port C1 bits
  test    laser_control,shutter   ;update for shutter
  jnz     shutter_closed
shutter_open:
  and     al,all-shutter_ctrl
  jmp     shutter_str
shutter_closed:       ;shutter closed if shutter not 0
  or      al,shutter_ctrl
shutter_str:
  out     dx,al
  ret

;
; AX = 0604:  set laser turn-off delay count
;
```

```
set_off_count:
  mov     bx,[bp].p_bx            ;set off tick count
  mov     off_delay_count,bx
  ret

;
; AX = 0605:  set laser turn-on delay count
;
set_on_count:
  mov     bx,[bp].p_bx            ;set on tick count
  mov     on_delay_count,bx
  ret

;
; AX = 0606:  set laser home position
;
set_home_pos:
  mov     bx,[bp].p_bx            ;home X-position in BX
  mov     HomeXpos,bx
  mov     cx,[bp].p_cx            ;home Y-position in CX
  mov     HomeYpos,cx
  ret
page

;-------------------------------------------------
;
; AH = 07:  control background task's execution
;
; only functional when Block Mode is on (laser_control
    bit 2)
;
; AL    EXECUTION CONTROL ESTABLISHED
; 00    execute command block
; 01    execute and clear command block
; 02    clear command block
;
;-------------------------------------------------
```

```
exec_ctrl_tbl label word
  dw      exec
  dw      exec_clear
  dw      clear
exec_ctrl_tbl_end equ $-exec_ctrl_tbl

bkgnd_exec_ctrl:
  test    current_status,ls_open  ;continue only if
            channel open
  jnz     exec_ctrl1
  mov     return_code,err_not_open
  ret
exec_ctrl1:
  test    laser_control,block_mode   ;  and in block
mode
  jnz     exec_ctrl2
  mov     return_code,err_not_block
exec_ctrl2:
  mov     ax,[bp].p_ax
  mov     di,bcmd_queue_end
  xor     ah,ah
  shl     ax,1
  cmp     ax,exec_ctrl_tbl_end
  jc      exec_call
  mov     return_code,err_bad_param  ;invalid function
  ret
  ;
  ;  call appropriate function
  ;
exec_call:
  mov     bx,ax
  call    word ptr cs:[bx + offset exec_ctrl_tbl]
  ret

;
; AX = 0700:  execute command block
;
```

```
; Execute command queue entries from last Clear
Operation
; to the last command entered.
;
exec:
  mov     bx,exec_queue_end    ;check for full exec
queue
  mov     si,bx
  add     bx,exec_entry_len
  cmp     bx,exec_queue_limit
  jb      exec1
  mov     bx,0          ;wraparound
exec1:
  cmp     bx,exec_queue_start ;full if indexes are equal
  jne     exec2
  mov     return_code,err_exec_limit
  ret
exec2:
  mov     ax,clear_pointer
  mov     [si],ax     ;establish block to execute:
  add     si,2    ;  from clear_pointer (set by clear
  mov     ax,bcmd_queue_end  ;  routine) to current end
of
  mov     [si],ax                 ;  command queue
  mov     exec_queue_end,bx       ;
  ret

;
; AX = 0701:  execute and clear command block
;
; Execute command queue entries from last Clear
Operation
; and then Clear.
;
exec_clear:
  call    exec                    ;set up exec block
  cmp     return_code,0
```

```
  je      clear          ;fall into clear if no error
  ret

;
; AX = 0702:  clear command block
;
; Reset the start of the command block for future
Execute
     Operations.
;
clear:
  mov     ax,bcmd_queue_end       ;set clear pointer to
             current end
  mov     clear_pointer,ax        ;  of command queue
  ret
page

;-------------------------------------------------
;
; AH = 08:  background task's redraw control
;
; AL    REDRAW CONTROL ESTABLISHED
; 00    set multi-pass step periods
; 01    set redraw count
; 02    set redraw delay
; 03    mark position
; 04    perform multi-pass redraw
;
;-------------------------------------------------
redraw_ctrl_tbl label word
  dw      add_set_periods
  dw      add_set_redraw_count
  dw      add_set_redraw_delay
  dw      add_mark_position
  dw      add_redraw
redraw_ctrl_tbl_end equ $-redraw_ctrl_tbl
```

```
redraw_ctrl:
  test    current_status,ls_open  ;continue only if
             channel open
  jnz     redraw_ctrl1
  mov     return_code,err_not_open
  ret
redraw_ctrl1:
  test    laser_control,block_mode  ;  and in block mode
  jnz     redraw_ctrl2
  mov     return_code,err_not_block
redraw_ctrl2:
  call    get_size      ;check for near-full queue
  cmp     ax,bcmd_queue_limit
  jb      redraw_ctrl3
  mov     return_code,err_queue_full
  ret
redraw_ctrl3:
  mov     ax,[bp].p_ax
  add     al,5                ;token val = subcmd # + 5
  mov     di,bcmd_queue_end
  mov     es:[di],al     ;store token
  inc     di
  mov     ax,[bp].p_ax
  xor     ah,ah
  shl     ax,1
  cmp     ax,redraw_ctrl_tbl_end
  jc      redraw_call
  mov     return_code,err_bad_param  ;invalid function
  ret
  ;
  ;  call appropriate function
  ;
redraw_call:
  mov     bx,ax
  call  word ptr cs:[bx + offset redraw_ctrl_tbl]
  ret
```

```
;
; AX = 0800:  add set multi-pass step periods to command
queue
;
; Add the set multi-pass step periods token and the
variable
; size argument to the Background Command Queue
;
; ES:BX points to step periods array:
; byte   0 - number of step periods (byte)
; 1,2 - step period pass 0     (word)
; 3,4 - step period pass 1     (word)
; etc.
;
add_set_periods:
  push  ds
  mov   ds,[bp].p_es  ;ES:BX points to step periods
array
  mov   si,[bp].p_bx
  cld                                ;for movsw, DS:SI+ ->
ES:DI+
  mov     cl,ds:[si]     ;get # step periods, word count
  mov     ch,0
  mov     es:[di],cl
  inc     si
  inc     di
  cmp     cx,10                      ;max of 10 passes
  jbe     add_set_periods1
  mov     cx,10
add_set_periods1:
  mov     bx,cx
  shl     bx,1                       ;compute laser command
size
  add     bx,set_periods_size
  rep movsw
  pop     ds
  jmp     end_store
```

```
;
; AX = 0801:  add set redraw count to command queue
;
; Add the set redraw count token and argument to the
; Background Command Queue
;
add_set_redraw_count:
  mov     bx,[bp].p_bx
  mov     es:[di],bx
  mov     bx,set_redraw_count_size
  jmp     end_store

;
; AX = 0802:  add set redraw delay to command queue
;
; Add the set redraw delay token and argument to the
; Background Command Queue
;
add_set_redraw_delay:
  mov  bx,[bp].p_bx
  mov  es:[di],bx
  mov  bx,set_redraw_delay_size
  jmp  end_store

;
; AX = 0803:  add mark position to command queue
;
; Add the mark position token to the Background Command
Queue
;
add_mark_position:
  mov     bx,mark_size
  jmp     end_store

;
; AX = 0804:  add redraw to command queue
;
```

```
; Add the redraw token to the Background Command Queue
;
add_redraw:
   mov     bx,redraw_size
  jmp     end_store

soft_fns endp

endcode label far

LaserCode ends

  end
^Z
```

```
'=====================================================
MATERIAL.BAS
' 3D Systems Stereolithography System Software
'=====================================================
'
' MATERIAL.BAS
' SLA-1 Material Manager Software
'
'-----------------------------------------------------
'
' History:
' 12/01/87  Ver 2.30  first release
' 12/14/87  Ver 2.40  handles line widths min and max,
    computes MSA
' 12/17/87  Ver 2.41  FN zero conditions handled
(divide,
    LOG)
' 1/08/88  Ver 2.50  relabeled Calc MSA due to MSA
concept
    problems
' 1/25/88  Ver 2.60  version update only
' 3/09/88  Ver 2.80  added copyright notice
' 4/09/88  Ver 3.00  production release
'
'-----------------------------------------------------

  DIM SP[20],LH[20],WMIN[20],WMAX[20]

  VersionId$ = "3.00"
  TRUE = -1
  FALSE = 0
  ScanFlag% = FALSE

  SingleBar$ = STRING$(45,196)
  PowerFile$         = "POWER.LST"
  LastMaterialFile$  = "MATERIAL.LST"
```

```
  COLOR 15
MatAsk:
  CLS
  PRINT SingleBar$
  PRINT "  MATERIAL, Material Manager"
  PRINT "  Version "+VersionId$
  PRINT "  Copyright (C) 1988 by 3D Systems, Inc."
  PRINT "  3D Systems Laser Stereolithography System"
  PRINT SingleBar$
  PRINT
  IF LEN(MaterialFile$)>0 THEN PRINT "     Material
File: ";MaterialFile$
  PRINT
  PRINT "      1. Load Material Data"
  PRINT "      2. View Material Data"
  PRINT "      3. Input New Material Data"
  PRINT "      4. Layer Thickness Information"
  PRINT "      Q. Quit"
  PRINT
  PRINT "      Option: ";
MatAskLoop:
  A$ = UCASE$(INKEY$)
  IF LEN(A$)=0 THEN MatAskLoop
  PRINT A$;
  IF A$="1" THEN LoadMaterial
  IF A$="2" THEN ViewMaterial
  IF A$="3" THEN InputMaterial
  IF A$="4" THEN LayerInfo
  IF A$="Q" THEN END

'-------------------------------------------------------

LoadMaterial:
  CLS
  PRINT "Available Material Files:"
  PRINT
  ON ERROR GOTO NoMaterialFiles
```

```
    FILES "*.MAT"
    GOTO GetMaterialAsk
NoMaterialFiles:
    PRINT "- no material files in working directory -"
    RESUME GetMaterialAsk
GetMaterialAsk:
    PRINT
    INPUT "Name of data file to read: ",MaterialFile$
    IF LEN(MaterialFile$)=0 THEN MatAsk
    MaterialFile$ = UCASE$(MaterialFile$)
    IF INSTR(MaterialFile$,".")=0 THEN MaterialFile$ =
MaterialFile$ + ".MAT"
    ON ERROR GOTO CantReadMaterialFile
    GOSUB GetMaterialFileData
    OPEN LastMaterialFile$ FOR OUTPUT AS #9
    PRINT #9,MaterialFile$
    CLOSE #9
    GOTO MatAsk

GetMaterialFileData:
    OPEN MaterialFile$ FOR INPUT AS #7
    ON ERROR GOTO 0
    INPUT #7,MaterialLaserPower
    INPUT #7,NumberOfPairs%
    FOR I% = 1 TO NumberOfPairs%
      INPUT #7,SP(I%)
      INPUT #7,LH(I%)
      INPUT #7,WMIN(I%)
      INPUT #7,WMAX(I%)
      SP(I%) = LOG(SP(I%))/LOG(10)
    NEXT I%
    CLOSE #7
    CALL Regression(LH(),LHSlope,LHYint,LHRval)
    CALL Regression(WMIN(),WMINSlope,WMINYint,WMINRval)
    CALL Regression(WMAX(),WMAXSlope,WMAXYint,WMAXRval)
    RETURN
```

```
CantReadMaterialFile:
  PRINT "Can't read material file ";MaterialFile$
  CLOSE #7
  RESUME GetMaterialAsk

'-----------------------------------------------------
InputMaterial:
  CLS
  PRINT "INPUT MATERIAL DATA"
  PRINT
  PRINT
  INPUT "Name of Material Data File: ",MaterialFile$
  MaterialFile$ = UCASE$(MaterialFile$)
  IF LEN(MaterialFile$)=0 THEN MatAsk
  IF INSTR(MaterialFile$,".")=0 THEN MaterialFile$ =
     MaterialFile$ + ".MAT"
  PRINT
  INPUT "Enter Material Test Laser Power Reading (mW):
    ",MaterialLaserPower
  INPUT "How many Step Period/Line Height data pairs?
    ",NumberOfPairs%
  PRINT
  ON ERROR GOTO 0
  FOR I% = 1 TO NumberOfPairs%
  PRINT "Data pair #";I%;":  ";
    INPUT "SP = ",SP(I%)
    LOCATE CSRLIN-1,30
    INPUT " LH = ",LH(I%)
    IF SP(I%)=0 OR LH(I%)=0 THEN MatAsk    ' abort
    SP(I%)=LOG(SP(I%))/LOG(10)             ' convert to
     log(sp)
    LOCATE CSRLIN-1,45
    INPUT "WMIN = ",WMIN(I%)
    LOCATE CSRLIN-1,60
    INPUT "WMAX = ",WMAX(I%)
  NEXT I%
  OPEN MaterialFile$ FOR OUTPUT AS #7
```

```
    PRINT #7,MaterialLaserPower
    PRINT #7,NumberOfPairs%
    FOR I% = 1 TO NumberOfPairs%
      PRINT #7,10^SP(I%)
      PRINT #7,LH(I%)
      PRINT #7,WMIN(I%)
      PRINT #7,WMAX(I%)
    NEXT I%
    CLOSE #7
    OPEN LastMaterialFile$ FOR OUTPUT AS #9
    PRINT #9,MaterialFile$
    CLOSE #9
    CALL Regression(LH(),LHSlope,LHYint,LHRval)
    CALL Regression(WMIN(),WMINSlope,WMINYint,WMINRval)
    CALL Regression(WMAX(),WMAXSlope,WMAXYint,WMAXRval)
    GOTO MatAsk

'-------------------------------------------------

ViewMaterial:
    CLS
    PRINT "Data from Material File: ";MaterialFile$
    PRINT
    PRINT
    PRINT "Material Test Laser Power Reading: ";
    PRINT USING "###.##";MaterialLaserPower
    PRINT
    PRINT NumberOfPairs%;"Material Data Pairs:"
    PRINT
    FOR I% = 1 TO NumberOfPairs%
      PRINT USING "##. ";I%;
      PRINT USING "  SP = ####.##";10^SP(I%);
      LOCATE ,20
      PRINT USING "  LH = ####.##";LH(I%);
      LOCATE ,35
      PRINT USING "  WMIN = ####.##";WMIN(I%);
      LOCATE ,52
```

```
    PRINT USING "  WMAX = ####.##";WMAX(I%)
  NEXT I%
  PRINT
  PRINT "  LH:  Slope =";
  PRINT USING "#####.##";INT(LHSlope*1000)/1000;
  PRINT "   Y-Intercept =";
  PRINT USING "#####.##";INT(LHYint*100)/100;
  PRINT "   R-Value =";
  PRINT USING "#####.##";INT(LHRval*1000)/1000
  PRINT "WMIN:  Slope =";
  PRINT USING "#####.##";INT(WMINSlope*1000)/1000;
  PRINT "   Y-Intercept =";
  PRINT USING "#####.##";INT(WMINYint*100)/100;
  PRINT "   R-Value =";
  PRINT USING "#####.##";INT(WMINRval*1000)/1000
  PRINT "WMAX:  Slope =";
  PRINT USING "#####.##";INT(WMAXSlope*1000)/1000;
  PRINT "   Y-Intercept =";
  PRINT USING "#####.##";INT(WMAXYint*100)/100;
  PRINT "   R-Value =";
  PRINT USING "#####.##";INT(WMAXRval*1000)/1000
  PRINT
  INPUT "Press Enter to continue... ",A$
  GOTO MatAsk

'-----------------------------------------------------

LayerInfo:
  CLS
  PRINT "LAYER THICKNESS INFORMATION"
  PRINT
  PRINT
  INPUT "Desired max layer thickness (mils): ",T$
  T = VAL(T$)    ' to simulate cure depth
  IF T=0 THEN GOTO MatAsk
  PRINT
  PRINT
```

```
  WMIN = FN GetLog(T+6,WMINSlope,WMINYint)
  WMAX = FN GetLog(T+6,WMINSlope,WMINYint)
'  MSA = 180/3.1415926535*(2*T/(WMIN+WMAX))    ' wrong
'  IF MSA<0 THEN MSA=0
'  IF MSA>89.999 THEN MSA=90
  PRINT USING "WMIN = ####.##";WMIN;
  PRINT USING "   WMAX = ####.##";WMAX
'  PRINT USING "   MSA = ###.##";MSA    don't show yet
  PRINT
  INPUT "Press Enter to continue... ",A$
  GOTO MatAsk

'-----------------------------------------------------

SUB Regression(Array(1),Slope,Yint,Rval)
  SHARED NumberOfPairs%,SP()
  XY        = 0
  SUMX      = 0
  SUMY      = 0
  SUMY2     = 0
  XSQUARED = 0
  IF NumberOfPairs%<1 THEN
    Slope = 0
    Yint  = 0
    Rval  = 0
    GOTO EndRegSub
  END IF
  FOR I% = 1 TO NumberOfPairs%
    XY = XY + SP(I%) * Array(I%)
    SUMX = SUMX + SP(I%)
    SUMY = SUMY + Array(I%)
    SUMY2 = SUMY2 + Array(I%)^2
    XSQUARED = XSQUARED + SP(I%)^2
  NEXT I%
  Denom = XSQUARED - (SUMX^2) / NumberOfPairs%
  IF Denom < 1E-5 THEN Slope = 0_
  ELSE Slope = (XY - (SUMX*SUMY)/NumberOfPairs%) / Denom
```

```
  Yint = SUMY/NumberOfPairs% - Slope *
SUMX/NumberOfPairs%
  Denom = SUMY2 - SUMY^2 / NumberOfPairs%
  IF Denom < 1E-5 THEN Rval = 0_
  ELSE Rval = SQR(((Yint * SUMY + Slope * XY) - SUMY^2/
    NumberOfPairs%) / Denom)
EndRegSub:
END SUB

DEF FN GetLog(DataPoint,Slope,Yint)
  IF DataPoint > 0 THEN
    FN GetLog = Slope * LOG10(DataPoint) + Yint
  ELSE
    FN GetLog = 0
  END IF
END DEF

'=====================================================
'
'  UTILITY FUNCTIONS AND PROCEDURES
'
'-----------------------------------------------------
'
' Removes all double spaces in CmdLine$
'
RemoveDoubleSpaces:
  WHILE INSTR(CmdLine$,"  ")>0
    I = INSTR(CmdLine$,"  ")
    CmdLine$ = left$(CmdLine$,I) + mid$(CmdLine$,I+2,99)
  WEND
  RETURN

'
'  Find any char of Char$ in Main$ starting at index
Start
'  Returns LEN(Main$)+1 if no character of Char$ in
Main$
```

```
'
DEF FN FindStr(Start%,Main$,Char$)
  LOCAL I%,J%,MIN%
  MIN% = LEN(Main$) + 1
  FOR I%=1 to LEN(Char$)
    J% = INSTR(Start%,Main$,MID$(Char$,I%,1))
    IF (J% > 0) AND (J% < MIN%) THEN MIN% = J%
  NEXT I%
  FN FindStr = MIN%
END DEF

'
'  Make a string from a number without a leading space
'
DEF FN StrNoSp$(Num)
  LOCAL A$
  A$=STR$(Num)
  IF LEFT$(A$,1)=" " THEN A$=RIGHT$(A$,LEN(A$)-1)
  FN StrNoSp$ = A$
END DEF

'
'  Pick out a number from a string
'  starting at Index and going to the next space or
comma
     or end-of-string
'
SUB UnitValInt(Storage&,Text$,Index%)
  Storage& = VAL(MID$(Text$,Index%))
  Index% = FN FindStr(Index%,Text$," ,") + 1
END SUB

SUB UnitVal(Storage,Text$,Index%)
  Storage = VAL(MID$(Text$,Index%))
  Index% = FN FindStr(Index%,Text$," ,") + 1
END SUB
```

```
'
'  Pick out text from a string
'  starting at Index and going to the next comma or
     end-of-string
'
SUB UnitStr(Storage$,Text$,Index%)
  LOCAL I%
  I% = FN FindStr(Index%,Text$," ,")
  Storage$ = MID$(Text$,Index%,I%-Index%)
  Index% = I% + 1
END SUB

'
'  Remove excess spaces around string
'
DEF FN NoSp$(Text$)
  LOCAL S%,E%
  IF LEN(Text$)>0 THEN
    S% = 1
    WHILE MID$(Text$,S%,1)=" "
      INCR S%
    WEND
    E% = LEN(Text$)
    WHILE E%>S%+1 AND MID$(Text$,E%,1)=" "
      DECR E%
    WEND
    FN NoSp$ = MID$(Text$,S%,E%-S%+1)
  ELSE
    FN NoSp$ = ""
  END IF
END DEF

'
'  Can't open file.  Report this to operator.
'
CantOpen:
  CALL SystemError("Can't open file "+FileName$)
```

```
  END

'-------------------------------------------------------
'
'  Show error and store it on disk
'
SUB SystemError(Text$)
  PRINT Text$
  OPEN "C:\SYSTEM.ERR" FOR OUTPUT AS #9
  PRINT #9,"PREPARE: ";Text$
  CLOSE #9
END SUB
^Z
```

```
{=======================================================
MENU.PAS
   3D Systems Stereolithography System Software
 =======================================================

   MENU.PAS
   SLA-1 Menu Software

 -------------------------------------------------------
   History:
   11/12/87  Ver 2.30  first Beta software release
   11/20/87  Ver 2.31  ENTER for Exit option fix
   11/23/87  Ver 2.32  pauses for Turbo Basic error
                       messages
   11/24/87  Ver 2.33  converted from Turbo Basic to
Turbo
                       Pascal SHELL's no longer eat up
64K
                       or main memory Consolidated
                       Profiler & Calibrator
                       Added Materials Manager to
Utility
                       Menu Added Change Directory to
Dos
                       Menu Alt-F9 View Option History
                       added
   12/02/87  Ver 2.34  fixed date format
   12/06/87  Ver 2.35  Using BEAM instead of CALIB and
                       PROFILE
   12/12/87  Ver 2.40  resets Text Mode when returning
                       from Shell
   12/16/87  Ver 2.41  clearing of screen for pause
fixed
                       prevent double pause (esp for
                       directory)
   12/17/87  Ver 2.42  revitalize cursor before running
                       Shells
```

```
  12/24/87  Ver 2.43  better cursor on/off routines
                      'X' input at ExitSystem verify
                      ignored
   1/08/88  Ver 2.50  TESTPART program access added
   1/25/88  Ver 2.60  version update only
   2/12/88  Ver 2.61  1900 + year in resetting system
                      date
   3/09/88  Ver 2.80  added copyright notice
                      installed new chronometer system
   4/04/88  Ver 3.00  added menu changes
   4/08/88  Ver 3.01  SUPER renamed to BUILD


-------------------------------------------------------}

{$R-,S+,I+,D+,T-,F-,V+,B-,N-,L+ }
{$M 4096,0,0 }

program MenuProgram;
uses Crt,Dos;

type
  str4    = string[4];
  str10   = string[10];
  str16   = string[16];
  str40   = string[40];
  str80   = string[80];
  str100 = string[100];
  HistoryType = record
    Menu,Option: Byte;
    mon,day,hour,min: Byte;
  end;
  DateType = record
    year : Word;
    month: Word;
    day  : Word;
  end;
```

```
  TimeStampType = record
    year : Word;
    month: Word;
    day  : Word;
    hour : Word;
    minu : Word;
    sec  : Word;
  end;
  YearType = (NotLeapYear,LeapYear);

var
  Fnkey,CmdChr,key,key2: Char;
  IOerr,CmdVal,ordkey: Integer;
  Row,Menu,Indent: Integer;
  Mon,Day,Year,Hour,Minu,Sec,WeekDay: Integer;
  WhenToExecute,ChronoOffset: array[1..32] of LongInt;
  WhatToExecute: array[1..32] of str40;
  code,i,ChronoPtr: Integer;
  PausedFlag,ErrorFlag,NoInterval: Boolean;
  Line,CurrentDate,CurrentTime: str80;
  temp,SingleBar,DoubleBar: str100;
  CurrentDir: str40;
  cmd: str16;
  regs: Registers;
  SystemFile,ChronoFile,ErrorFile: Text;
  MenuPos: array[1..3] of Integer;
  OptionHistory: array[1..19] of HistoryType;
  StartHistory,EndHistory: Integer;
  MinTimeStamp: TimeStampType;
  CurrentMinutes: LongInt;
  UpdateFile: File of Char;

const
  MSDOS = $21;
  VersionId = '3.01';
  ComSpec = 'C:\COMMAND.COM';
  SystemFileName = 'C:\SLA1.SYS';
```

```
  ErrorFileName = 'C:\SYSTEM.ERR';
  ChronoFileName = 'C:\CHRONO.SYS';
  sp80 = '
                                    ';
  DayOfWeek: String[21] = 'SunMonTueWedThuFriSat';
  Zel: array[1..12] of Integer =
(1,4,3,6,1,4,6,2,5,0,3,5);
  MainExecs: array[1..8] of str10 =
    ('XFER','MERGE','GRAPH','PREPARE','BUILD','','','');
  UtilityExecs: array[1..8] of str10 =

('POWER','GOBEAM','ZSTAGE','MATERIAL','EDIT','TESTPART',
'','');
  DosExecs: array[1..8] of str10 =
    ('CHKDSK','','DIR/W','QD2','','','','');
  MenuMax: array[1..3] of Integer = (8,7,6);
  MenuColor: array[1..3] of Integer = (12,11,10);
  MaxHistory = 19;
  Cr = ^M;
  DaysInMonth: array[NotLeapYear..LeapYear,1..12] of
Word
    = ((31,28,31,30,31,30,31,31,30,31,30,31),
       (31,29,31,30,31,30,31,31,30,31,30,31));
  DayTable: array[NotLeapYear..LeapYear,1..13] of Word
    = ((0,31,59,90,120,151,181,212,243,273,304,334,365),

(0,31,60,91,121,152,182,213,244,274,305,335,366));

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
```

```
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure ResetScreen;
  begin
    TextColor(White);
    TextBackground(Black);
    TextMode(C80);
    CursorOff;
  end;

function upper(text: String): String;
var
  i: Integer;
  temp: String;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
   upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): str80;
var
  temp: str80;
```

```
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function spc(len: Integer): String;
begin
  spc := copy(sp80,1,len);
end;

procedure beep;
begin
  sound(1000);
  delay(100);
  NoSound;
end;

function IntStr(num: Integer): str10;
var
  temp: str10;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

procedure ReadCommandLine(var line: str80);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
```

```
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Compute day-of-week using Harry's modified Zeller
Algorithm
}
function Zeller(month,day,year: Integer): Integer;
begin
  if month<3 then dec(year);
  Zeller := (Zel[month] + day + year + (year shr 2)) mod
     7;
end;

{
  Read time directly from clock chip
```

```
  works only on ATs, 286s, 386s; does not work on
standard
   PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;     { wait for update to finish
(then
     there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;     { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;     { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;     { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;     { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $09;     { get year }
                Year := port[$71];
                port[$70] := $0A;
```

```
                    if (port[$71] and $80)=0 then good :=
                     true;
                  end;
                end;
              end;
            end;
          end;
        end;
      end;
      Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
       from BCD }
      Minu := (Minu shr 4) * 10 + (Minu and $0F);
      Hour := (Hour shr 4) * 10 + (Hour and $0F);
      Day  := (Day  shr 4) * 10 + (Day  and $0F);
      Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
      Year := (Year shr 4) * 10 + (Year and $0F);
      WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the
   current time
}
procedure GetTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
```

```
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

{
  Reset Computer System's Date and Time from Hardware
   Clock
}
procedure SetSystemDateTime;
begin
  GetTime;
  SetDate(1900+Year,Mon,Day);
  SetTime(Hour,Minu,Sec,0);
end;

{
  Julian-Cummulative-Minute algorithms
}
function Julian(date: DateType): Word;
var
  YearLookup: YearType;
begin
  with date do begin
    if (year mod 4) = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
```

```
    Julian := 365*(year-1900) +
DayTable[YearLookup,month] +
      day + ((year-1901) div 4) - 32142 {1/1/88};
  end;
end;

procedure UnJulian(JulNum: Word;var date: DateType);
var
  YearLookup: YearType;
  offset,j,lastj: LongInt;
begin
  with date do begin
    inc(JulNum,32142);   {1/1/88}
    year := 1900;
    j := 0;
    lastj := 0;
    while j <= JulNum do begin
      lastj := j;
      if year mod 4 = 0 then inc(j,366)
        else inc(j,365);
      inc(year);
    end;
    dec(year);   { back off a year }
    if year mod 4 = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
    offset := JulNum - lastj;
    month := 1;
    j := 0;
    lastj := 0;
    while j <= offset do begin
      lastj := j;
      j := DayTable[YearLookup,month];
      inc(month);
    end;
    dec(month,2);
    day := offset - lastj + 1;
  end;
```

```
end;

function AbsMinutes(TimeStamp: TimeStampType): LongInt;
var
  date: DateType;
begin
  with TimeStamp do begin
    date.year  := year;
    date.month := month;
    date.day   := day;
    AbsMinutes := LongInt(Julian(date)) * 1440 + (hour *
     60 + minu);
  end;
end;

procedure AbsTimeStamp(Minutes: LongInt;var TimeStamp:
   TimeStampType);
var
  date: DateType;
  DayMinutes: Word;
begin
  DayMinutes := Minutes mod 1440;
  UnJulian(Minutes div 1440,date);
  with TimeStamp do begin
    year  := date.year;
    month := date.month;
    day   := date.day;
    hour  := DayMinutes div 60;
    minu  := DayMinutes mod 60;
  end;
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
var
```

```
  s,e: Integer;
begin
  s := 1;
  while (Text[s] in ['"',' ']) and (s < length(Text)) do
   inc(s);
  e := length(Text);
  while (Text[e] in ['"',' ']) and (e > s) do dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

function InChildProcess: Boolean;
begin
  InChildProcess := (Mem[0:99 shl 2] and $80)>0;
end;

procedure SetChildProcess;
begin
  Mem[0:99 shl 2] := $80;
end;

procedure ClearChildProcess;
begin
  Mem[0:99 shl 2] := $00;
end;

procedure CheckForOptions(var line: str80);
var
  i: Integer;
begin
  line := upper(line);
  i := pos('/NOINTERVAL',line);  { check for
    /NOINTERVAL
 }
  if i>0 then begin
    NoInterval := true;
    delete(line,i,11);
  end;
```

```
end;

procedure Pause;
begin
  if not PausedFlag then begin
    gotoxy(1,25);
    write('Press Enter to Continue... ');
    readln(Input);
    PausedFlag := true;
  end;
end;

function Desc(menu,line: Integer): str80;
var
  index: Integer;
begin
  index := menu * 10 + line;
  case index of
    11: Desc := 'Transfer Files Between this Computer
and
       Another over Ethernet';
    12: Desc := 'Merge Separately-Sliced Object Files
       Together';
    13: Desc := 'Graph an Object in either Faceted or
       Sliced form';
    14: Desc := 'Prepare to Make a Part and Set Up
Various
       Parameters';
    15: Desc := 'Run the Stereolithography Part-Builder
       Program';
    16: Desc := 'Go to the Utility Menu';
    17: Desc := 'Go to the Disk Operating System Menu';
    18: Desc := 'Exit Stereolithography System
       Completely';
    21: Desc := 'Control the Power-Up or Down Sequence
of
       Laser Equipment';
```

```
      22: Desc := 'Laser Beam Profile and Calibration
         Operations';
      23: Desc := 'Move the Elevator Up and Down';
      24: Desc := 'View, Alter, and Add Data for Vat
         Materials';
      25: Desc := 'Enter the Editor to Modify the Contents
         of a File';
      26: Desc := 'Run the Test Part Utility Program';
      27: Desc := 'Return to the Main Menu';
      31: Desc := 'Show Hard Disk Statistics, including
Free
         Space Remaining';
      32: Desc := 'Change to a Different File Directory';
      33: Desc := 'Show the Directory for the Current
         Directory';
      34: Desc := 'Enter the Quick-DOS Utility to Move
Files
         and Change Directories';
      35: Desc := 'Enter a DOS Shell, Allowing You to Run
         Other Programs';
      36: Desc := 'Return to the Main Menu';
      else Desc := '';
    end;
  end;

  {-------------------------------------------------------}

  procedure DrawMain;
  begin
    writeln('                                        MAIN
  MENU');
    TextColor(White);
    writeln;
    gotoxy(20,whereY);
    writeln(copy(SingleBar,1,40));
    writeln;
    gotoxy(Indent,whereY);
```

```
  writeln('1. Network       ');
  gotoxy(Indent,whereY);
  writeln('2. Merge');
  gotoxy(Indent,whereY);
  writeln('3. View  ');
  gotoxy(Indent,whereY);
  writeln('4. Prepare');
  gotoxy(Indent,whereY);
  writeln('5. Build');
  gotoxy(Indent,whereY);
  writeln('6. Utility Menu');
  gotoxy(Indent,whereY);
  writeln('7. DOS Menu');
  gotoxy(Indent,whereY);
  writeln('X. Exit');
end;

procedure DrawUtility;
begin
  writeln('                              UTILITY MENU');
  TextColor(White);
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  writeln;
  gotoxy(Indent,whereY);
  writeln('1. Power Sequencer');
  gotoxy(Indent,whereY);
  writeln('2. Beam Analysis');
  gotoxy(Indent,whereY);
  writeln('3. Elevator Mover');
  gotoxy(Indent,whereY);
  writeln('4. Material Manager');
  gotoxy(Indent,whereY);
  writeln('5. Edit a File');
  gotoxy(Indent,whereY);
  writeln('6. Make Test Parts');
```

```
  gotoxy(Indent,whereY);
  writeln('X. Return to Main Menu');
end;

procedure DrawDos;
begin
  writeln('                                        DOS
MENU');
  TextColor(White);
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  writeln;
  gotoxy(Indent,whereY);
  writeln('1. Disk Usage Stats');
  gotoxy(Indent,whereY);
  writeln('2. Change Directory');
  gotoxy(Indent,whereY);
  writeln('3. Directory Listing');
  gotoxy(Indent,whereY);
  writeln('4. Quick DOS Utility');
  gotoxy(Indent,whereY);
  writeln('5. DOS Shell');
  gotoxy(Indent,whereY);
  writeln('X. Return to Main Menu');
end;

procedure DrawMenuScreen;
begin
  SetSystemDateTime;
  ClrScr;
  TextColor(White);
  writeln(DoubleBar);
  writeln(' SLA-1  3D SYSTEMS STEREOLITHOGRAPHY SYSTEM
V'
    + VersionId);
```

```
  writeln(' All Programs and Screens Copyright (C) 1988
by
                3D Systems, Inc.');
  writeln(DoubleBar);
  writeln;
  TextColor(MenuColor[Menu]);
  case Menu of
    1: DrawMain;
    2: DrawUtility;
    3: DrawDos;
  end;
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  Row := whereY;
  writeln;
  writeln(' Select an Option either by Using the Arrows
           and Pressing Enter');
  write  ('  or by Pressing the Corresponding Number');
end;

function ErrorOnScreen: Boolean;
var
  BaseAddr: Word;
  ScanError: Boolean;
  i: Integer;
begin
  if LastMode=7 then BaseAddr := $B000
    else BaseAddr := $B800;
  i := 0;
  ScanError := true;
  while (i<4000) and ScanError do begin
   if upcase(chr(Mem[BaseAddr:i]))='E' then
      if (upcase(chr(Mem[BaseAddr:i+2]))='R') and
        (upcase(chr(Mem[BaseAddr:i+4]))='R') then
         ScanError := false;
    inc(i,2);
```

```
  end;
  ErrorOnScreen := not ScanError;
end;

{
  SHELL program, check for existence of error
  If error, pause and wait for Enter before returning to
   menu
}
procedure Shell(Prog,Options: str80);
begin
  PausedFlag := false;
  SetSystemDateTime;
  CursorOn;
  Exec(Prog,Options);
  CursorOff;
  {$I-}
  assign(ErrorFile,ErrorFileName);
  reset(ErrorFile);
  {$I+}
  ErrorFlag := (IOresult=0);
  if ErrorFlag then begin
    close(ErrorFile);
    Pause;
    {$I-}
    erase(ErrorFile);
    {$I+}
  end else if ErrorOnScreen then Pause;
end;

{-------------------------------------------------------}

{
  Option History routines
}
procedure AddHistory(text: str10);
begin
```

```
    OptionHistory[EndHistory].Menu := Menu;
    OptionHistory[EndHistory].Option := CmdVal;
    OptionHistory[EndHistory].mon := Mon;
    OptionHistory[EndHistory].day := Day;
    OptionHistory[EndHistory].hour := Hour;
    OptionHistory[EndHistory].min := Minu;
    inc(EndHistory);
    if EndHistory > MaxHistory then EndHistory := 1;
    if StartHistory = EndHistory then begin
      inc(StartHistory);
      if StartHistory > MaxHistory then StartHistory := 1;
    end;
  end;

  procedure ShowHistory;
  begin
    ClrScr;
    writeln('Menu System Recent Option History:');
    writeln;
    writeln('Menu  Option  Name            Date   Time');
    writeln('----  ------  --------------  -----  -----');
    i := StartHistory;
    while i<>EndHistory do with OptionHistory[i] do begin
      write('  ',Menu,'      ',Option,'    ');
      case Menu of
        1: cmd := MainExecs[Option];
        2: cmd := UtilityExecs[Option];
        3: cmd := DosExecs[Option];
      end;
      write(cmd,spc(16-length(cmd)));
      writeln(PadSp(mon),'/',LZero(day),'
',PadSp(hour),':',LZero(min));
      inc(i);
      if i > MaxHistory then i := 1;
    end;
    writeln;
    write('Press any key to continue... ');
```

```
  key := ReadKey;
  DrawMenuScreen;
end;

{
  Show Menu Item Description
}
procedure ShowDescription;
begin
  gotoxy(Indent-2,9+MenuPos[Menu]);
  TextColor(Yellow);
  write(chr(16));
  gotoxy(1,24);
  write(spc(79));
  gotoxy((78-length(Desc(Menu,MenuPos[Menu]))) div
2,24);
  TextColor(Black);
  TextBackground(MenuColor[Menu]);
  write(' ',Desc(Menu,MenuPos[Menu]),' ');
  TextColor(White);
  TextBackground(Black);
end;

procedure GetCurrentMinutes;
begin
  MinTimeStamp.year  := 1900 + Year;
  MinTimeStamp.month := Mon;
  MinTimeStamp.day   := Day;
  MinTimeStamp.hour  := Hour;
  MinTimeStamp.minu  := Minu;
  CurrentMinutes := AbsMinutes(MinTimeStamp);
end;

{
  chronometry routines
}
function IntParse: Integer;
```

```
var
  num: Real;
  IntNum,code,code2: Integer;
begin
  if length(temp)=0 then IntNum := 0
  else begin
    while copy(temp,1,1) = ' ' do delete(temp,1,1);
    val(temp,num,code);
    if code<>0 then val(copy(temp,1,code-1),num,code2);
    IntNum := trunc(num);
    if code=0 then temp := ''
      else temp := copy(temp,code+1,99);
  end;
  IntParse := IntNum;
end;

procedure ReadChronoFile;
var
  ChronoLine: String;
  TimeStamp: TimeStampType;
  DayOffset,HourOffset,MinuOffset: Integer;
begin
  ChronoPtr := 0;
  assign(ChronoFile,ChronoFileName);
  {$I-}
  reset(ChronoFile);
  {$I+}
  if IOresult<>0 then exit;
  while not eof(ChronoFile) do begin
    readln(ChronoFile,ChronoLine);
    if (ChronoLine <> '') and (ChronoLine[1] <> '!')
then
      begin
      inc(ChronoPtr);
      temp := ChronoLine;
      with TimeStamp do begin
        month := IntParse;
```

```
          day   := IntParse;
          year  := 1900 + IntParse;
          hour  := IntParse;
          minu  := IntParse;
          WhenToExecute[ChronoPtr] :=
AbsMinutes(TimeStamp);
        end;
        DayOffset  := IntParse;
        HourOffset := IntParse;
        MinuOffset := IntParse;
        ChronoOffset[ChronoPtr] := (LongInt(DayOffset) *
24
          + HourOffset) * 60
          + MinuOffset;
        i := pos('!',temp);
        WhatToExecute[ChronoPtr] :=
NoQuotes(copy(temp,1,i-1));
(*
{}      writeln('ChonoPtr=',ChronoPtr);
{}      writeln('WTE=',WhenToExecute[ChronoPtr]);
{}      writeln('CO =',ChronoOffset[ChronoPtr]);
{}      writeln('WTE="',WhatToExecute[ChronoPtr],'"');
{}      key := ReadKey;
*)
    end;
  end;
  close(ChronoFile);
end;

procedure UpdateChronoFile(index: Integer;NewTimeStr:
  str16);
var
  i,j: Integer;
  ChronoLine: String;
  OldUpdateFilePos,UpdateFilePos: LongInt;
begin
  i := 0;
```

```
 assign(ChronoFile,ChronoFileName);
 {$I-}
 reset(ChronoFile);
 {$I+}
 if IOresult<>0 then exit;
 UpdateFilePos := 0;
 while not eof(ChronoFile) do begin
   OldUpdateFilePos := UpdateFilePos;
   readln(ChronoFile,ChronoLine);
   inc(UpdateFilePos,length(ChronoLine)+2);
   if (ChronoLine <> '') and (ChronoLine[1] <> '!')
then
     begin
     inc(i);
     if i = index then begin
       close(ChronoFile);
       assign(UpdateFile,ChronoFileName);
       {$I-}
       reset(UpdateFile);
       {$I+}
       if IOresult<>0 then exit;
       Seek(UpdateFile,OldUpdateFilePos);
       for j:=1 to length(NewTimeStr) do
         write(UpdateFile,NewTimeStr[j]);
       close(UpdateFile);
       exit;
     end;
   end;
  end;
  close(ChronoFile);
end;

procedure ChronoExecute(index: Integer);
var
  NextTime: TimeStampType;
  tmon,tday,tyear,thour,tminu: str4;
begin
```

```
  gotoxy(1,24);
  ClrEol;
  TextColor(Black);
  TextBackground(White);
  gotoxy(25,24);
  write('   PLEASE WAIT MOMENTARILY   ');
  TextColor(White);
  TextBackground(Black);
  Exec(ComSpec,'/C '+WhatToExecute[Index]);
  while keypressed do key := ReadKey;   { flush out
  keyboard buffer }
  ResetScreen;
  DrawMenuScreen;
  ShowDescription;
  GetCurrentMinutes;
  WhenToExecute[Index] := CurrentMinutes +
ChronoOffset[Index];
  AbsTimeStamp(WhenToExecute[Index],NextTime);
  with NextTime do begin
    tmon  := PadSp(month);
    tday  := LZero(day);
    tyear := LZero(year-1900);
    thour := LZero(hour);
    tminu := LZero(minu);
    UpdateChronoFile(index,tmon+'/'+tday+'/'+tyear+'
     '+thour+':'+tminu);
  end;
end;

{
  Get Input for Menu, handle scrolling
}
procedure HelpAsk;
var
  i: Integer;
  LineChange,Leave: Boolean;
begin
```

```
Leave := false;
LineChange := true;
repeat
  if LineChange then ShowDescription;
  repeat
    GetTimeStamp;
    GetDir(0,CurrentDir);
    gotoxy(2,5);
    write(CurrentDir);
    gotoxy(67,5);
    write(CurrentDate);
    gotoxy(71,6);
    write(CurrentTime);
    GetCurrentMinutes;
    for i:=1 to ChronoPtr do
      if WhenToExecute[i] <= CurrentMinutes then
         ChronoExecute(i);
  until keypressed;
  LineChange := false;
  CmdChr := upcase(ReadKey);
  if CmdChr = chr(0) then FnKey := ReadKey
    else FnKey := chr(0);
  if CmdChr in [#0,'1'..'9','X',Cr] then begin
    LineChange := true;
    gotoxy(Indent-2,9+MenuPos[Menu]);
    write(' ');
    if (CmdChr=Cr) and (MenuPos[Menu]<MenuMax[Menu])
      then
      CmdChr := chr(ord('0')+MenuPos[Menu]);
    if (CmdChr='X') or ((CmdChr=Cr) and
(MenuPos[Menu]=MenuMax[Menu])) then
      CmdChr := chr(ord('0')+MenuMax[Menu]);
    case FnKey of
      'P': if MenuPos[Menu] >= MenuMax[Menu] then
        MenuPos[Menu] := 1
           else inc(MenuPos[Menu]);
      'H': if MenuPos[Menu] <= 1 then MenuPos[Menu]
```

```
            := MenuMax[Menu]
                 else dec(MenuPos[Menu]);
          'p': ShowHistory;
        end;
        val(CmdChr,CmdVal,code);
        if (CmdVal>0) and (CmdVal<=MenuMax[Menu]) then
begin
          Leave := true;
          MenuPos[Menu] := CmdVal;
        end;
      end;
    until Leave;
    case Menu of
      1: cmd := MainExecs[CmdVal];
      2: cmd := UtilityExecs[CmdVal];
      3: cmd := DosExecs[CmdVal];
    end;
    if length(cmd)>0 then AddHistory(cmd);
end;

{-------------------- UTILITY STUFF -------------------}

procedure UtilityMenu;
begin
  repeat
    Menu := 2;
    DrawMenuScreen;
    HelpAsk;
    ClrScr;
    if CmdVal<7 then begin
      Shell(ComSpec,'/C '+UtilityExecs[CmdVal]);
      ResetScreen;
    end;
  until CmdVal = MenuMax[2];
end;

{----------------- DOS STUFF -----------------}
```

```
procedure DiskUsage;
begin
  writeln('       DISK USAGE STATISTICS');
  writeln(copy(SingleBar,1,34));
  writeln;
  Shell(ComSpec,'/C '+DosExecs[1]);
  Pause;
end;

procedure DirList;
begin
  Shell(ComSpec,'/C '+DosExecs[3]);
  Pause;
end;

procedure ChangeDir;
var
  NewDir: str40;
begin
  CursorOn;
  ClrScr;
  writeln('Current directory is ',CurrentDir);
  writeln;
  writeln;
  writeln('You can specify a new directory using the
    following rules:');
  writeln;
  writeln('If you wish to move to a subdirectory, just
    type in the');
  writeln('subdirectory''s name.');
  writeln;
  writeln('If you want to move up a directory level,
just
    enter .. (two periods).');
  writeln;
  writeln('If you are going to specify a completely new
    directory path, start');
```

```
    writeln('your entry with a \ (backslash).');
    writeln;
    writeln;
    write('Enter new directory specification: ');
    readln(NewDir);
    {$I-}
    ChDir(NewDir);
    if IOresult<>0 then begin
      writeln;
      writeln('Can''t find directory.');
      writeln;
      write('Press any key to continue... ');
      key := ReadKey;
    end;
    {$I+}
    CursorOff;
  end;

  procedure QDos;
  begin
    Shell(ComSpec,'/C '+DosExecs[4]);
  end;

  procedure DosShell;
  begin
    CursorOn;
    ClrScr;
    SetChildProcess;
    writeln('To return to the Dos Menu, use the EXIT
      command.');
    writeln(copy(SingleBar,1,48));
    Exec(ComSpec,'');
    ResetScreen;
    ClearChildProcess;
  end;

  procedure DosMenu;
```

```
begin
  repeat
    Menu := 3;
    DrawMenuScreen;
    HelpAsk;
    ClrScr;
    case CmdVal of
      1: DiskUsage;
      2: ChangeDir;
      3: DirList;
      4: QDos;
      5: DosShell;
    end;
  until CmdVal = MenuMax[3];
end;

{----------------------- MAIN STUFF -----------------}

procedure ExitSystem;
begin
  CursorOn;
  ClrScr;
  writeln('Dos Menu Option 5 allows you to issue DOS
   commands and run other');
  writeln('programs, then return to the Menu System
using
   the EXIT command.');
  writeln;
  writeln('This option will cause you to leave the
   Stereolithography System');
  write  ('completely.  Are you sure you want to do
   this? ');
  repeat
    key := upcase(ReadKey);
  until key<>'X';
  writeln(key);
  if key='Y' then begin
```

```
    SetSystemDateTime;
    Halt(0);
  end;
  CursorOff;
end;

procedure MainMenu;
begin
  ResetScreen;
  repeat
    Menu := 1;
    DrawMenuScreen;
    HelpAsk;
    ClrScr;
    if CmdVal=MenuMax[1] then ExitSystem
    else if CmdVal<6 then begin
      Shell(ComSpec,'/C '+MainExecs[CmdVal]);
      ResetScreen;
    end else if CmdVal=6 then UtilityMenu
    else if CmdVal=7 then DosMenu;
  until 0=1;  { forever }
end;

{
  read SLA-1 System File
}
procedure ReadSystemFile;
label
  NoMoreSystemFile;
begin
  {$I-}
  assign(SystemFile,SystemFileName);
  reset(SystemFile);
  {$I+}
  if IOresult<>0 then begin
    writeln('No system file.  Stopping.');
    halt(1);
```

```
 end;
 repeat
   readln(SystemFile,Line);
 until eof(SystemFile) or ((length(Line)>0) and
  (Line[1]<>'!'));
 if not eof(SystemFile) then begin
   Line := NoQuotes(Line);
   CheckForOptions(Line);
   i := 1;
   while (i<2) and not eof(SystemFile) do begin
     Line := '';
     while (length(Line)=0) or (Line[1]='!') do begin
       if eof(SystemFile) then goto NoMoreSystemFile;
       readln(SystemFile,Line);
     end;
     Line := upper(Line);
     case i of
       1: begin
{ ????? }
          end;
     end;
     inc(i);
   end;
 end;
NoMoreSystemFile:
 close(SystemFile);
end;

{  -  -  -  -  -  }

begin
 CheckBreak := false;
 if InChildProcess then begin
   writeln('You are currently in a DOS Shell.');
   writeln('Please use EXIT to return to the Menu
    System.');
    halt(2);
```

```
  end;
  Indent := (80-20) div 2;
  NoInterval := false;
  for i:=1 to 3 do
    MenuPos[i] := 1;
  StartHistory := 1;
  EndHistory := 1;

  SingleBar := many(#196,79);
  DoubleBar := many(#205,79);

  SetSystemDateTime;
  {$I-}
  assign(ErrorFile,ErrorFileName);
  erase(ErrorFile);
  IOerr := IOresult;
  {$I+}
  ReadSystemFile;
  ReadChronoFile;
  ReadCommandLine(Line);
  CheckForOptions(Line);
  MainMenu;
end.
^Z
```

```
MERGE.PCT
{=========================================================
  3D Systems Stereolithography System Software
=========================================================

MERGE.PAS
SLA-1 Merge Sliced Files Software
---------------------------------------------------------

Compile with Turbo Pascal Version 4.0

File name line syntax:

<file name>/Z<offset>/X<offset>/Z<offset>
  /N<blocks>/O<blocks> <file name>...
  /OUTFILE<outfile>/RES<resolution>/ZSPC
     <z-spacing>/THICK<thickness>

where <file name> is a Slice input file
       <offset>    is a + or - X,Y or Z-axis offset for
                   that input file
       <blocks>    contains the identifiers for the
                   blocks to skip for that input file
                  separated by commas
       <outfile>   follows all input file specs and is
                   the name of the part output file to
                  store the parsed data

Up to 10 Slice data files can be combined.
---------------------------------------------------------

Merge *.V vector output file format:   (file #2)

  L1,<layer A>
  <x1> <y1> <x2> <y2>
   :    :     ]
  <another block,<layer A>
```

```
  :    :
[L2,<layer A>
 <x1> <y1> <x2> <y2>
  :    :     ]
 <another block>,<layer A>
  :    :]
 L1,<layer B>
  :    :
 (EOF)
```

Critical Area Codes:

```
  XV = draw section cross-hatch vectors in box as rivets
  XI = ignore section cross-hatch vectors in box
  SV = draw up/down skin cross-hatch vectors in box
       as rivets
  SI = ignore up/down skin cross-hatch vectors in box
```

Recent History:

9/ 1/87	Ver 2.04	Version 2 released
9/ 3/87	Ver 2.07	Compiled by TBM (Modified Turbo Basic)
9/ 4/87	Ver 2.08	added SV, SI critical areas
9/17/87	Ver 2.10	only updated version number
9/25/87	Ver 2.20	new SLICE 19 block format supported
9/28/87	Ver 2.21	added NFUB suppport
9/30/87	Ver 2.25	only updated version number
10/26/87	Ver 2.27	Parse name changed to Merge *.P files now *.V files error file support
11/12/87	Ver 2.30	first Beta software release

```
11/20/87  Ver 2.31  added directory listing of *.SLI
                    files
          Ver 2.32  -
12/04/87  Ver 2.33  slice20 FUB FDB blocks
                    BLOCK.ID file support
12/14/87  Ver 2.40  input file names in *.L output
                    file pass along ! comment lines in
                    slice files
12/21/87  Ver 2.41  second !MERGE line contains start,
                    end, # of layers
                    BOX file noted when found, not if
                    not found
                    scans !SLICE comments for Res,
                    ZSpacing, Thick
 1/11/88  Ver 2.50  BLOCK.ID looked for in \3DSYS too
 1/22/88  Ver 2.54  makes Range instead of Layer
                    Control File as default
                    Layer Control File made if
                    /LAYERCTRL option used
 1/25/88  Ver 2.60  generates both Layer and Range
                    Files (.L and .R)
 1/27/88            thickness input is in mils, not
                    millimeters
 1/27/88            thickness defaults to Z-Spacing
                    value
 2/10/88            converted from Turbo Basic to
                    Turbo Pascal 4.0 merge operation
                    runs 2.5 times faster than before
                    merge status box with layer and
                    range information
                    /O only and /N no block selections
                    improved
 2/12/88  Ver 2.61  EndLayer variable now LongInt
                    layer numeric output is type word,
                    not integer
 3/09/88  Ver 2.80  added copyright notice
```

```
   3/22/88  Ver 2.90  Create binary V file
                      Add file offsetting option
   4/08/88  Ver 2.91  Eliminate Critical Area Boxes
                      Add Min Max values to Headers
   4/09/88  Ver 3.00  Production
   4/15/88  Ver 3.01  Replace/Offset#,# (X and Y) with
                      /X# and /Y# to parallel the /Z#
                      otpion; also set RC<>0 for case of
                      merge file not found.

-----------------------------------------------------------}

uses Crt,Dos;

const
  VersionId= '3.01':
  sp80='
  ErrorFileName = C:\SYSTEM.ERR';
  BlockIdFileName = 'BLOCK.ID';
  HomeDirectory = 'C:\3DSYS\';
  Cr = ^M;
  StatusBoxLeft   = 42;
  StatusBoxRight  = 79;
  StatusBoxTop    = 10;
  StatusBoxBottom = 22;

type
  str6  = string[6];
  str16 = string[16];

var
  FirstVar: Byte; { must be first variable defined }
  CurrentZ,ZOffset: array [1..10] of LongInt;
  OnlyBlocks,NoBlocks: array [1..10] of String;
  BlockMnemonics: array [1..32] of str6;
  Blocks: array [1..100] of str6;
  ErrorFile: Text;
```

```
  InputFiles: array[1..10] of Text;
  code,NumPartFiles,InFile,NumCritVols: Integer;
  BlockMnemonicsCount,ZSpace,BlockCount,Layers: Integer;
  CmdLine,MergeLine,FirstFileName,Line,OutFileName:
String;
  SingleBar,LayerFileName,RangeFileName,VectorFileName:
    String;
  Resolution,ZSpacing,Thickness: Real;
  CurrentLayer,LastLayer,ZStart,ZEnd,ZVal: LongInt;
  Startlayer,EndLayer,MergeLayer: LongInt;
  LayerFile,RangeFile,VectorTextFile: Text;
  VectorBinaryFile: File
  BinaryOutput : Boolean;
  UpdateFile: File of Char;
  UpdateIndex: LongInt;
  HeaderType: Char;
  FileNum,Cmd,Blk: str6;
  FirstRange,NewRange,SlipBlock,NeedStep: Boolean;
  RangeLayerCount,RangeCount,UpdX,UpdY: Integer;
  OffsetX,OffsetY: Array [1..10] of LongInt;
  VectorCount : LongInt;
  VectorCountPtr : LongInt;
  rc : Integer;
  MinX,MaxX,MinY,MaxY : Word;
  LastVar: Byte; { should be last variable defined }

{-------------------------------------------------------

                    Utility Routines
-------------------------------------------------------}

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
```

```
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;
function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;
function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr :=temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr :=temp;
end;
  ;
function LongStr(num: LongInt): str16;
var
```

```
  temp: str16;
begin
  Str(num,temp);
  LongStr:=temp;
end;
function RealStr(num: Real;AfterDecimal: Integer):
String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal.temp);
  i :=1;
  while (i<length(temp)) and (temp[i]='  ') do inc(i);
  RealStr :=copy (temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc :=copy(sp80+sp80,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line :=line + '  ' + ParamStr(i);
end;

{
  Remove excess quoe marks around string
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
```

```
var
  s,e: Integer;
begin
  s :=1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e :=length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0) do
    delete(text,pos(' ',text),1);
end;

{
  Find any char of Chars in Main starting at index Start
  Returns length(Main)+1 if no character of Chars in
Main
}
function FindStr(Start: Integer;Main: string;Chars:
str16:
   Integer;
var
  i,j,min: Integer;
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
  for i:=1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
```

```
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  Code2: Integer;
  num:   Long Int;
begin
  Val(line,num,code);
  if Code <>0 then val(copy(line,1,code-1),num,code2);
  LongVal  :=num;
end;

function RealVal(line:String", var code:Integer): Real;
var
  code2: Integer;
  num: Real;
begin
  ofs := 0;
```

```
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end
  val(line,num,code);
  if code<>0 then val (copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  ReadVal := num;
end;

{
  Pick out a number from a string starting at Index and
  going to the next space or comma or end-of-string
}
procedure UnitValShortInt(var Storage: Integer;Text:
  String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure Unit ValInt(var Storage: LongInt;
  Text:String;var Index:Integer);
var
  i:integer;
begin
  Storage: = LongVal(Copy(Text,Index, 99),i);
  if i = 0 then Index: = length(Text)+1
    else inc(Index,i);
  end;

procedure UnitVal(varStorage;Real;Text:String;VarIndex:
  Integer);
var
```

```
  i:Integer;
begin
  Storage: = RealVal(copy(Text,Index,99)i);
  if i = 0 then Index := length(Text) +1
    else inc(Index,i);
end;

{
  Pick out text from a string starting at Index and
going
    to the next comma or end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
   Integer);
var
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

{
  Show error and store it on disk
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'MERGE: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
```

```
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{ Harry's ultimate Draw-A-Box on the text screen
    routine }
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1: Single Line all around
      2: Double Line all around
      3: Single Line horizontal, Double Line vertical
      4: Double Line horizontal, Single Line vertical
}
  const
                    {   Type        1    2    3    4   }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then begin)
```

```
    if y1=y2 then FullHorzBar := Many(HorzBar[BoxType],
    x2-x1+1)
      else FullHorzBar := Many (HorzBar[BoxType],
        x2-x1-1);
    end else FullHorzBar := '';
  if y1<>y2 then begin
    if x1<>x2 then write(UpperLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>X2 then write(UpperRightCorner[BoxType]);
end,
for i:=y1+1 to y2-1 do begin
  gotoxy(x1,i);
  write(VerBar[BoxType]);
  gotoxy(x2,i);
  write(VertBar[BoxType]);
end;

  gotoxy(x1,y2);
  if y1 = y2 then write (FullHorzBar)
  else begin
    if x1<>x2 then write(LowerLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(LowerRightCorner[BoxType]);
  end;
end;

procedure SetUpStatusBox;
begin
    DrawBox(StatusBoxLeft,StatusBoxTop,StatusBoxRight,
    StatusBoxBottom,1);
  DrawBox(StatusBoxLeft+1,StatusBoxTop+2,StatusBox
    Right-1,
    StatusBoxTop+2,1);
  DrawBox(StatusBoxLeft+1,StatusBoxBottom-2,Status
    BoxRight-1);
    StatusBoxBottom-2,1);
end;
```

```
{=====================================================}

procedure FindFirstLayers;
var
  i,dummy: Integer;
  GoodLine: Boolean;
begin
{ determine first layer number for each slice file }
  MergeLayer := 99999;
  for InFile := 1 to NumPartFiles do begin
    if not eof(InputFiles[InFile]) then begin
      GoodLine := false;
      repeat
        readln(InputFiles[InFile],Line);
        if (Line = '') or (Line[1] in ['!','*']) then
          begin
          writeln(LayerFile,Line);
          writeln(RangeFile,Line);
          if copy(Line,1,6) = '!SLICE' then begin
            i := pos('-RES ',Line);
            if (i>0) and (Resolution<=0) then
              Resolution :=  RealVal(copy
                (Line,i+5,99,dummy);
            i := pos('-ZS ',Line);
            if (i>0) and (ZSpacing<=0) then
              ZSpacing := RealVal(copy
                (Line,i+4,99),dummy);
            i := pos('-THICK ', Line);
            if (i>0) and (Thickness<=0) then
              Thickness := RealVal(copy
                (Line,i+7,99),dummy);
          end
        end else GoodLine := true;
      until GoodLine or eof(InputFiles[InFile]);
      HeaderType := Line[1];
      Zval := LongVal(copy(Line,3,99),dummy) + ZOffset
        [(InFile];
```

```
        CurrentZ[InFile] := Zval;
        if Zval < MergeLayer THEN MergeLayer := Zval;
            end;
          end;
          StartLayer := MergeLayer;
        end;

procedure GetParameters;
var
  MmFlag: Boolean;
  dummy: Integer;
  DefaultThickness: Real;
  temp: String;
begin
  writeln;
  if Resolution <= 0 then repeat
    write('Slice -res resolution value? ');
    readln(Line);
    Resolution := RealVal(Line,dummy);
  until Resolution>0;
  if ZSpacing <= 0 then begin
    write('Slice -zs layer spacing? ');
    readln(Line);
    ZSpacing := RealVal(Line,dummy);
  end;
  MmFlag := false;
  if Thickness <= 0 then begin
    DefaultThickness := int(ZSpacing * 1000 /
Resolution);
    write('Layer thickness in mils [',RealStr
      (DefaultThickness,0),']? ');
    readln(line);
    if Line = '' then Thickness := DefaultThickness
    else begin
      if pos('MM',upper(Line))>0 then MmFlag := true;
      Thickness := RealVal(Line,dummy);
    end;
```

```
  end;
  if not MnFlag then Thickness := Thickness * 0.0254;
    { convert to mm }
  temp := '* ' + RealStr(Resolution,3) + ',' + RealStr
    (ZSpacing,3) + ',' + RealStr(Thickness,3);
  writeln(LayerFile,temp);
  writeln(RangeFile,temp);
end;

{-------------------------------------------------------}

procedure AddBlock;
var
  i: Integer;
begin
  i := 1;
  while (i <= BlockCount) and (Blocks[i] <> Blk) do
    inc(i);
  if i > BlockCount then begin
    inc(BlockCount);
    Blocks[BlockCount] := Blk;
  end;
end;

procedure OutputRange;
var
  i: Integer;
begin
  gotoxy(updX,UpdY);
  write(LastLayer:5,' ',RangeLayerCount:5);
  NewRange := true;
  writeln(RangeFile,ZStart,',',LastZSpace,',',ZSpace);
  for i:=1 to BlockCount do
    writeln(RangeFile,Blocks[i]);
  inc(RangeCount);
end;
procedure StartNewRange;
```

```
begin
  ZStart := MergeLayer;
  BlockCount := 0;
  NeedStep := true;
end;

procedure MakeControl;
var
  LayerDiff: LongInt;
begin
  writeln(LayerFile,MergeLayer,',',Blk);
  if Layers =1 then begin
    ZStart :=MergeLayer;
    CurrentLayer := MergeLayer;
    NeedStep := true;
    AddBlock;
  end else begin
    if CurrentLayer = MergeLayer then AddBlock
    else begin
      CurrentLayer := MergeLayer;
      if NeedStep then begin
        NeedStep := false;
        ZSpace := MergeLayer - LastLayer;
        AddBlock;
      end else begin
        LayerDiff := MergeLayer - LastLayer;
        if LayerDiff <> ZSpace then begin
          OutputRange;
          StartNewRange;
          AddBlock;
        end else AddBlock;
      end;
    end;
  end;
end;

{-------------------------------------------------------}
```

```
procedure NewLayer;
var
  i: Integer;
begin
  inc(Layers);
  EndLayer := MergeLayer;
  if NewRange then begin
    if FirstRange then begin
      LastLayer := MergeLayer;
      FirstRange := false;
    end;
    writeln;
    write(Cr,RangeCount+1:2,'.',LastLayer:5,' - ');
    UpdX := whereX;
    UpdY := whereY;
    write(LastLayer:5,'    1 leyers');
    NewRange := false;
    RangeLayerCount := 1;
  end else begin
      gotoxy(UpdX,UpdY);
      inc(RangeLayerCount);
      write(LastLayer:5, ' ',RangeLayerCount:5);
  end;
  Blk := '#TOP';
  MakeControl;
end;

{-------------------------------------------------------
}

procedure OutBinBlock(BlockStr : String);
  var
    i : Integer;
    BlockChar : Char;
  begin
    i:=1;
```

```
    while (i<=BlockMnemonicsCount) and
BlockMnemonics[i]<>
      BlockStr) do Inc(i);
    if i>BlockMnemonicsCount then BlockChar := #$FF
    else BlockChar := Char (i);
    BlockWrite(VectorBinaryFile,BlockChar,1);
  end;

procedure OutBinSet(MergeSetStr : String);
  var
    MergeSetNum : Byte;
  begin
    Val(MergeSetStr,MergeSetNum,rc);
    if rc<>0 then MergeSetNum := $FF
    BlockWrite(VectorBinaryFile,MergeSetNum,1);
  end;

procedure OutBinNum(LongNum : LongInt);
  var
    WordNum : Word;
  begin
    if LongNum>65535 then LongNum := $FFFF;
    WordNum :=LongNum;
    BlockWrite(VectorBinaryFile,WordNum2);
  end;

procedure OutVectorLine(line : String;FileNum :
  Integer);
  var
    x,y,i : Integer;
    TempWord : Word;
  begin
    x:=1;
    for i:=1 to 4 do begin
      while (x<=Length(line)) and (line[x]=' ') do
Inc(x);
      if x>Length(Line) then begin
```

```
          writeln;
          SystemError('Error: point not found in merge
           file');
          writeln('File ',FileNum,' "',line,'"');
          Halt(1);
        end;
        y:=Pos(' ',Copy(line,x,Length(line(-x+1));
        if y=0 then y := Length(line)+1
        else y:=y+x-1;
        Val(Copy(line,x,y-x),TempWord,rc);
        if rc<>0 then begin
          writeln;
          SystemError('Error in point in merge
            file line');
          writeln('File ',FileNum,' "',line,'"');
          Halt(1);
        end;
        x:=y;
        if (i=1) or (i=3) then begin
          TempWord:=TempWord+OffsetX[FileNum];
          if TempWord<MinX then MinX:=TempWord;
          if TempWord>MaxX then MaxX:=TempWord;
        end else begin
          TempWord:=TempWord+OffsetY[FileNum];
          if TemWord<MinY then MinY:=TempWord;
          if TempWord>MaxY then MaxY:=TempWord;
        end;
        if BinaryOutput then BlockWrite(VectorBinaryFile,
          TempWord,2)
        else begin
          if i=1 then write(VectorTextFile,TempWord)
          else write(VectorTextFile,' ',TempWord);
        end;
      end;
      if Not BinaryOutput then writeln(VectorTextFile);
      Inc(VectorCount);
    end;
```

```
procedure WriteVectorCount;
  begin
    if VectorCount>0 then begin
      seek(VectorBinaryFile,VectorCountPtr);
      BlockWrite(VectorBinaryFile,VectorCount,4);
      VectorCount := 0;
      seek(VectorBinaryFile,FileSize(VectorBinaryFile));
    end;
  end;

{-----------------------------------------------------}

procedure PrintHeaders;
begin
(*  write(Cmd,FileNum,' '); *)
  { update vector file }
  if BinaryOutput then begin
    WriteVectorCount;
    OutBinBlock(Cmd);
    OutBinSet(FileNum);
    OutBinNum(MergeLayer);
    VectorCountPtr := FileSize(VectorBinaryFile);
    BlockWrite(VectorBinaryFile,VectorCount,4);
  end else
writeln(VectorTextFile,Cmd,FileNum,',',MergeLayer);
  { update layer and range files }
  Blk := Cmd + FileNum;
  MakeControl;
end;
```

```
{-----------------------------------------------------}
procedure ReadSlices;
var
  i,j,c: Integer;
  select,temp: String;
begin
  FirstRange  := true;
  NewRange    := true;
  repeat
    NewLayer;
    for InFile := 1 to NumPartFiles do begin
      FileNum := IntStr(InFile);
      { process rest of Z block & other blocks }
      SkipBlock := false;
      if not(eof(InputFiles[InFile])) and
        (CurrentZ[InFile]
        =MergeLayer) then begin
        Cmd := HeaderType;
        PrintHeaders;
      end;
      while not(eof(InputFiles[InputFiles[InFile])) and
        (CurrentZ[InFile]=MergeLayer) do begin
        repeat
        readln(InputFiles[InFile],Line);
        Line[length(Line) + 1] := #0;
        if copy(Line,1,1) = '!' then begin
          writeln(LayerFile,Line);
          writeln(RangeFile,Line);
        end;
      until copy(Line,1,1)<>'!';
{
  process slice block
}
        i := 0;
        temp := copy(Line,1,2);
        if (temp = 'Z ') or (temp = 'L ') then i :=1
        else if length(temp)=1 then i := pos(temp,
```

```
        'ZXYISHVABJCDEFGK');
      if i=0 then begin
        i := 1;
        while (i <= BlockMnemonicsCount) and
      ((copy(Line,1,length(BlockMnemonics[i])) <>
        BlockMnemonics[i]) or not (Line[length(Block
        Mnemonics[i]))+1] in [#0,' ']))
          or not (Line[length(BlockMnemonics[i])+1]
            in [#0,' ']))
            do inc(i);
        if i > BlockMnemonicsCount then i := 0;
      end;
      if i > 0 then begin
        SkipBlock := false;
        c := FindStr(1,line,' ');
        Cmd := copy(Line,1,c-1);
        if i=1 then begin    { handle Z block special)
          Zval := LongVal(copy(Line,c+1,9),code) +
           ZOffset[InFile];
          CurrentZ[InFile] := Zval;
        end else begin
          select := ',' + cmd + '.';
        if (pos(select,NoBlocks[InFile]) > 0) or
          ((length(OnlyBlocks[InFile]) > 0) and
            (pos(select,OnlyBlocks[InFile])=0)) then
              begin
          SlipBlock := true;
        end else PrintHeaders;
      end;
    end else if not SlipBlock then OutVectorLine
      (Line,InFile);
end; {while}
end; {for}
Blk := '#BTM';
MakeControl;
LastLayer  := MergeLayer;
MergeLayer := 99999;
```

```
    for InFile := 1 to NumPartFiles do
      if not eof(InputFiles[InFile]) then
        if CurrentZ[InFile] < MergeLayer then MergeLayer
          := CurrentZ[InFile];
  until MergeLayer = 99999;
  if BinaryOutput then WriteVectorCount;
end;

procedure Update(FileName: String);
var
  i: Integer;
  St: String;
begin
  assign(UpdateFile,FileName);
  reset(UpdateFile);
  Seek(UpdateFile,UpdateIndex);
  St := WordStr(StartLayer);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex +16);
  St  := WordStr(End Layer);
  St  := St + spc (5-length(St));
  for i:1 to 5 do write(UpdateFile,St[i];
  Seek(UpdateFile,UpdateIndex + 16 + 17);
  St := WordStr(Layers);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 60);
  St  :=  WordStr(MinX);
  St  :=  St + spc(5-length (St));
  for i:1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex 60 + 16);
  St := WordStr(MaxX);
  St := St + spc (5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 60 +43);
  St := WordStr(MinY);
```

```
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i];
  Seek(UpdateFile,UpdateIndex + 60 + 59);
  St := WordStr(MaxY);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  close(UpdateFile);
end;

{-------------------------------------------------------}

procedure MergeFiles;
begin
{ determine output file prefix }
  if OutFileName = '' then begin
    write('Output File Name Prefix [',FirstFileName,']:
');
    readln(OutFileName);
    if OutFileName = '' then OutFileName :=
FirstFileName;
    OutFileName := upper(OutFileName);
  end;

{ open up output files, existing files are overwritten }
  LayerFileName    := OutFileName + '.L';
  RangeFileName    := OutFileName + '.R';
  VectorFileName   := OutFilename + '.V';
  assign(LayerFile,LayerFileName);
  rewrite(LayerFile);
  writeln(layerFile,'!MERGE: -FILES ',MergeLine);
  writeln(LayerFile,'!MERGE: -STARTLAYER XXXXX -ENDLAYER
    XXXXX -NUMLAYERS XXXXX');
  writeln(LayerFile,'!MERGE: -MIN X      XXXXX -MAX X
    XXXXX');
  writeln(LayerFile,'!MERGE: -MIN Y      XXXXX -MAX Y
    XXXXX');
  UpdateIndex := length(MergeLine) + 17 + 20;   { points
```

```
    to StartLayer value}
  assign(RangeFile,RangeFileName);
  rewrite(RangeFile);
  writeln(RangeFile,'!MERGE: -FILES ',MergeLine);
  writeln(RangeFile,'!MERGE: -STARTLAYER XXXXX
    -ENDLAYER  XXXXX -NUMLAYERS XXXXX');
  writeln(RangeFile,!MERGE: -MIN X       XXXXX -MAX X
      XXXXX');
  writeln(RangeFile,!MERGE: -MIN Y       XXXXX -MAX Y
      XXXXX');

  Layers := 0;
  MinX:=65535;
  MinY:=65535;
  MaxX:=0;
  MaxY:=0;
  FindFirstLayers;
  GetParameters;
  if BinaryOutput then begin
    assign(VectorBinaryFile,VectorFileName);
    rewrite(VectorBinaryFile, 1);
    rc:=0;
    BlockWrite(VectorBinaryFile,rc,1);
  end else begin
    assign(VectorTextFile,VectorFileName);
    rewrite(VectorTextFile);
  end;

window(StatusBoxLeft+1,StatusBoxTop+1,StatusBoxRight-1,
  StatusBoxBottom-1);
ClrScr;
window(1,1,80,25);
SetUpStatusBox;
gotoxy(StatusBoxLeft+2,StatusBoxTop+1);
write('Different Z-Spacing Rangers:');
window(StatusBoxLeft+2,StatusBoxTop+3,StatusBoxRight-2,
  StatusBoxBottom-3);
```

```
ReadSlices;

if BinaryOutput then close(VectorBinaryFile)
else close(VectorTextFile);
OutputRange;
close(LayerFile);
close(RangeFile);
Update(LayerFileName);
Update(RangeFileName);
window(1,1,80,25);
gotoxy(StatusBoxleft+2,StatusBoxBottom-1);
write(RangeCount,' range');
if RangeCount>1 then write('s');
write(', ',Layers,' layer');
if Layers>1 then write('s');
write(' processed.');
end;

{-------------------------------------------------------}
procedure CheckForOptions;
var
  i,oi,j,k: Integer;
  FileName: String;
begin
  FirstFilename := '';
  CmdLine := upper(CmdLine);
  RemoveDoubleSpaces(CmdLine);
  i := pos ('/OUTFILE',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,8);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
    inc(i);
    j := FindStr(i,CmdLine,' /');
    OutFileName := copy(CmdLine,i,j-i);
    delete(CmdLine,oi,j-oi);
  end;
```

```
  i := pos('/RES',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,4);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
    inc(i);
    j := FindStr(i,CmdLine,' /');
    val(copy(CmdLine,i,j-i),Resolution,code);
    delete(CmdLine,oi,j-oi);
  end;
  i := pos('/ZSPC',CmdLine);
  if i>0 then begin
    ci := i;
    inc(i,5);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
    inc(i);
    j := FindStr(i,CmdLine,' /');
    val(copy(CmdLine,i,j-i),ZSpacing,code);
    delete(CmdLine,oi,j-oi);
  end;

i := pos('/ASCII',CmdLine);
BinaryOutput := True;
if i>0 then begin
  oi := i;
  inc(i,6);
  while (i<length(CmdLine)) and (CmdLine[i]=' ') do
  inc(i);
  j := FindStr(i,CmdLine,' /');
  BinaryOu ut := False;
  delete(C. Line,oi,j-oi);
end;

i := pos('/THICK',CmdLine);
if i>0 then begin
  oi := i;
  inc(i,6);
```

```
  while (i<length(CmdLine)) and (CmdLine[i]=' ') do
  inc(i);
  j := FindStr(i,CmdLine,' /');
  val(copy(CmdLine,i,j-i),Thickness,code);
  delete(CmdLine,oi,j-oi)
end;

  RemoveDoubleSpaces(CmdLine);
  MergeLine := CmdLine;

{ pick out slice file names and open them up}
  while CmdLine<>'' do begin
    i :=pos(' ',CmdLine);
    if i=0 then begin
      FileName := CmdLine;
      CmdLine := '';
    end else begin
      FileName := copy(CmdLine,1,i-1);
      CmdLine  := copy(CmdLine,i+1,255);
    end;
    inc(NumPartFiles);
{}
    i := pos('/X',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
do
       inc(i);
      j := FindStr(i,FileName,' /');
    {   val(copy(FileName,i,j-i),XOffset[NumPartFiles],
          code); }
      if i<j then OffsetX[NumPartFiles] := LongVal(copy
        (FileName, i,j-i),k);
      delete(FileName,oi,j-oi);
    end;
```

```
      i := pos('/Y',FileName);
      if i>0 then begin
        oi := i;
        inc(i,2);
        while (i<length(FileName)) and (FileName[i]=' ')
          do inc(i);
        j := FindStr(i,FileName,' /');
        if i<j then OffsetY[NumPartFiles] := LongVal(copy
          (FileName,i,j-i),k;
        delete(FileName,oi,j-oi);
      end;

      i := pos('Z/,FileName);
      if i=0 then i := pos('/L',FileName);
      if i>0 then begin
        oi := i;
        inc(i,2);
        while (i<length(FileName)) and (FileName[i]=' ')
do
         inc(i);
      j := FindStr(i,FileName,' /');
      val(copy(FileName,i,j-i),ZOffset[NumPartFiles],
        code);
      delete(FileName,oi,j-oi);
    end;
    i := pos('/N',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]='  ') do
       inc(i);
      j := FindStr(i,FileName,'/');
      NoBlocks[NumPartFiles] := ','+copy
        (FileName,i,j-i)+',';
      delete(FileName,oi,j-oi);
    end;
```

```
{**** OFFSET Option. Offsets given file by X,Y on output
  OffsetX[NumPartFiles] := 0;
  OffsetY[NumPartFiles] := 0;
  i := pos('/OFFSET',FileName);
      ifi>0 then begin
        oi :=i;
        inc(i,7);
        while (i<length(FileName)) and (FileName[i]='
          ') do inc(i);
        j := FindStr(i,FileName,'/');
        OffsetX[NumPartFiles] := LongVal(copy(FileName,
          i,j-i),k);
      if k<>0 then begin
        i :=i + k;
        if i<j then OffsetY[NumPartFiles] :=
LongVal(copy
          (FileName,i,j-i),k);
      end;
      delete(FileName,oi,j-oi);
    end; *****}

    i := pos('/O',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
        do inc(i);
      j := FindStr(i,FileName,'/');
      OnlyBlocks[NumPartFiles] := ','+copy(FileName,
        i,j-i,+',';
      delete(FileName,oi,j-oi);
    end;
    if FirstFileName = '' then FirstFileName :=
FileName;
    i := pos('.',FileName);
    if i=0 then FileName := FileName +'.SLI';
    assign(InputFiles[NumPartFiles],FileName);
```

```
    ($I-)
    reset(InputFiles[NumPartFiles]);
    if IOresult<>0 then begin
      writeln('Error: Can''t open file ',FileName);
      halt(1);
    end;
  end; {while}
  if NumPartFiles = 0 then halt(0);
end;

procedure ShowFiles(FileSpec: String);
var
  FileCount: Integer;
  DirInfo: SearchRec;
begin
  FileCount: Integer;
  FindFirst(FileSpec,AnyFile,DirInfo);
  if DosError<>0 then writeln('- none -')
  else begin
    while DosError=0 do begin
      if FileCount>0 then write(spc(11-((whereX-1)
        mod 10+1)));
      inc(FileCount);
      write(copy(DirInfo.Name,1,pos('.',DirInfo.Name)-
1));
      FindNext(DirInfo);
    end;
    writeln;
  end;
end;

procedure ProcessCommandLine;
begin
  ReadCommandLine(CmdLine);
  if CmdLine = '' then begin
    writeln('Directory Listing of Sliced Files:');
    writeln;
```

```
      ShowFiles('*.SLI');
      writeln;
      write('Slice File Names: ');
      readln(CmdLine);
      writeln;
    end;
    NumPartFiles := 0;
    CheckForOptions;
  end;

  procedure ReadBlockIdFile;
  var
    BlockIdFile: text;
    BlockIdLine: string;
    index,Group: Integer;
    Block: String;
  begin
    BlockMnemonicsCount := 0;
    {$I-}
    assign(BlockIdFile,BlockIdFileName);
    reset(BlockIdFile);
    if IOresult<>0 then begin

  assign(BlockIdFile,HomeDirectory+'\'+BlockIdFileName);
      reset(BlockIdFile);
      if IOresult<>0 then begin
        writeln('Can''t find Block Id file.');
        halt(2);
      end;
    end;
    {$I+}
    while not eof(BlockIdFile) do begin
      readln(BlockIdFile,BlockIdLine);
      if (length(BlockIdLine)>0) and (BlockIdLine[1]<>'!')

        then begin
        index := 1;
```

```
        UnitValShortInt(Group,BlockIdLine,index);
        UnitStr(Block,BlockIdLine,index);
        if Group=0 then begin
          inc(BlockMnemonicsCount);
          BlockMnemonics[BlockMnemonicsCount]
            := upper(Block);
        end;
      end;
      close(BlockIdFile);
end;

{------------------------MAIN PROGRAM-------------------
}

begin
  ExitProc := @ExitMerge;
  CheckBreak := true;
  FillChar(FirstVar,ofs(lastVar-Ofs(FirstVar),#0);
    { clear variables  }
  ClrScr;
  SingleBar := many(#196,26);
  centerln('MERGE, Sliced Files Merger');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  centerln(SingleBar);

  writeln;
  writeln;
  OutFileName := '';
  ReadBlockIdFile;
  ProcessCommandLine;
  MergeFiles;
  QuietSystemError('Success.');
end.
```

```
{=========================================================
  3D Systems Stereolithography System Software
=========================================================

  MONITOR.PAS
  SLA-1 Laser Beam Monitoring Program
---------------------------------------------------------
  Compile with Turbo Pascal Version 4.0
---------------------------------------------------------
  Recent History:
  3/14/88 Ver 2.80  first release
  4/09/88 Ver 3.00  production release
---------------------------------------------------------
program Monitor;
uses Crt,Dos;
type
  str6  = string[6];
  str10 = string[10];
  str16 = string[16];
  BeamProfileVarSensorType = record
    Xpos               : LongInt;  { inputs }
    Ypos               : LongInt;
    ScanStep           : Integer;
    BkgndThreshold     : Integer;
    InitialDelay       : Integer;
    RowToRowDelay      : Integer;
    ColToColDelay      : Integer;
    XYweight           : LongInt; { inputs }
    Xweight            : LongInt;
    TotalWeight        : LongInt;
    HiPoints           : Words;
    SaturationCount    : Integer;
    MaxSensorReading   : Integer;
    BeamMap            : array[0..30,0..30] of Integer;
    RowWeight          : array[0..30] of Word;
    ColWeight          : array[0..30] of Word;
  end;
```

```
BeamProfileVarType = array[1..3] of BeamProfile
  VarSensorType;
FindBeamVarSensorType = record
  Xval              : LongInt;  { inputs & updated }
  Yval              : LongInt;
  SpiralDelay       : Word;     { inputs }
  SpiralSideMax     : Integer;
  SpiralTries       : Integer;
  ScanStep          : Integer;
  BkgndThreshold    : Integer;
  NumCenterPasses   : Integer;
  Xavg              : Real; { inputs }
  Yavg              : Real;
end;
FindBeamVarType = array[1..3] of FindBeamVarSensorType;
FindMethodType = (none,NormalFind,Verify2MilHole);
const
  VersionId = '3.00';
  ParamFileName = 'BEAM.PRM'
  ErrorFileName = 'C:\SYSTEM.ERR';
  HomeDirectory = 'C:\3DSYS\';
  PowerHistoryFileName    = 'C:\HISTORY\POWER.MON';
  ProfileHistoryFileName  = 'C:\HISTORY\PROFILE.MON';
  Blank = '
  ColMax = 22;
  RowMax = 22;

  LaserAddr   = $300
  NumSensors  = 2;
  TimeOut     = 60000;

var
  FirstVar: Byte: { must be first variable defined }
  { beam arrays }
  Resistance,Temperature: array[0..100] of Real;
  { sensor arrays }
```

```
    SensorChannel,XSensorPos,YSensorPos,SensorBkgnd:
     array[1..3] of Word;
    Power,Bk: array[1..3] of Real;
    Xsp,Ysp: array[1..3] of Real;
    SensorCalib: array[1..3,1..4] of Real;
    LastField: array[1..4] of Integer;
    { miscellaneous }
    key: Char;
    Transparent,BeamFound: Boolean;
    CurrentDate,CurrentTime,OutFileName,SingleBar,CmdLine:
     String;
    XRest,YRest,TempChannel,TemCalPts: Word;
    i,PeakReading,Sensor,Select,SensorNum: Integer;
    GoodCount,Sec,Minu,Hour,Day,WeekDay,Mon,Year:  Integer;
    Xpos,Ypos: LongInt;
    Xaverage,Yaverage,TotWtAvg: Real;
    OldExitProc: Pointer;
    BeamProfileVariables: BeamProfileVarType;
    FindBeamVariables: FindBeamVarType;
    CalibDelays: array[1..3,1..3] of LongInt;
    ErrorFile,ParamFile,PowerHistoryFile,Profile
      HistoryFile: Text;
    LastVar: Byte; { should be last variable defined }

{-------------------------------------------------------
                    Utility Routines
-------------------------------------------------------}

procedure beep;
begin
  sound(1000);
  delay(300);
  NoSound;
end;

function upper(text: string): string;
var
```

```
  i: Integer;
  temp: string;
begin
  temp :=  text;
  for i := 1 to length(temp) do temp[i] := upcase(temp[i];
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp := temp+Ch;
  many := temp;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
```

```
begin
  Str(num,temp);
  Longstr := temp;
end;

function RealStr(num: Real;AfterDecimal: Integer): String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal,temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc := copy(blank,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
var
```

```
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;

function WordVal(line: String;var code: Integer): Word;
var
  code2: Integer;
  num: Word;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;

function LongVal(line:String;var code:Integer): LongInt;
var
  code2:  Integer;
  num:    LongInt;
begin
  val(line,num,code);
  if code<> then val(copy(line,1,1,code-1),num,code2);
  LongVal :=  num;
end;
```

```
function RealVal(line: String;var code: Integer): Real;
var
  ofs,code2: Integer;
  num: real;
begin
  ofs :=0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  RealVal := num;
end;

{
  Show error and store it on disk
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'MONITOR: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
end;

{
  Can't open file. Report this to operator.
}
procedure CantOpen(FileName: string);
```

```
begin
  SystemEror('Can''t open file '+FileName);
end;

{$F+}
procedure ExitMonitor;
{$F-}
begin
  ExitProc := OldExitProc;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := Intstr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;
```

```
{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
  standard PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;     { wwait for update to finish
      (then there's 244 us)  }
    if (port[$71] and $80)=0 then begin
      port[$70]:= $00;     { get seconds }
      Sec := port[$71]
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02; { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04; { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;   { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08; { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $09; { get year }
                Year := port[$71];
                port[$70] := $0A;
```

```
                    if (port[$71] and $80)=0 then good
                      := true;
                  end;
                end;
              end;
            end;
          end;
        end;
      end;
      Sec  := (Sec  shr 4) * 10 + (Sec  and $0F); { convert
                                                    from BCD }
      Minu := (Minu shr 4) * 10 + (Minu and $0F);
      Hour := (Hour shr 4) * 10 + (Hour and $0F);
      Day  := (Day  shr 4) * 10 + (Day  and $0F);
      Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
      Year := (Year shr 4) * 10 + (Year and $0F);
    end;

    {
      Get a time stamp consisting of the relative day
        and the current time
    }
    procedure GetTimeStamp;
    var
      tday,tmon,tyear,tsec,tminu,thour: str10;
      AmPm: Char;
    begin
      GetTime;
      tmon  := PadSp(Mon);
      tday  := LZero(Day);
      tyear := LZero(Year);
      CurrentDate := tmon+'/'+tday+'/'+year;
      tsec  := LZero(Sec);
      tminu := LZero(Minu);
      if Hour=0 then begin
        thour := '12';
        AmPm := 'A';
```

```
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end

function TimeStamp : String;
begin
  GetTimeStamp;
  TimeStamp := CurrentDate + '   ' + Current Time;
end;

function ReadLaserBoard(Channel: Integer): Integer;
var
  count := Timeout;
  ADC,i: Integer;
begin
  count := Timeout;
  port[LaserAddr+12] := Channel;
  ( must wait 10 us for EOS to go LO (286 @ 4us/loop) }
  i := 1;
  while i < 10 do inc(i);
  repeat  { then approx 110 us for conversion with
    signal D7 HI }
    dec (count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if PeakReading < ADC then PeakReading := ADC;
  if count = 0 then ReadLaserBoard := -1
    else ReadLaserBoard := ADC;
```

```
end;

procedure ReadTemperature(var Resist,Temper: Real);
var
  temp: Real;
  It,ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board
  use 1/T, LOG(R) function To linearize the interpolation
}
  ADC := ReadLaserBoard(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC);
    { from 20K resistor on JB Board }
  if ADC>214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] >
    Resist) do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It])); /
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) - ln(Resist)));
  end;
  Temper := Temper - 273.15;
end;

procedure println(text: String);
begin
  if not transparent then writeln(text);
end;
```

```
{-------------------------------------------------------
                BEAM SCANNING ROUTINES
-------------------------------------------------------}

                        ************
                        * MoveBeam *
                        ************

Routine to move mirrors to X,Y directly
Note: X,Y can be any kind of variable, we change to
     integers and get ADC% (for speed: later fixed addrs
      routines for each)
}
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 THEN POS := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
    { x is bit 0, y is bi
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInAxis(x: LongInt);
begin
  AxisMove(x,1);
end;

procedure MoveInYAxis(y: LongInt);
begin
  AxisMove(y,2)
end;

procedure MoveBeam(x,y: LongInt);
begin
  MoveInXAxis(x);
```

```
  MoveInYAxis(y);
end;

{
                    ******************
                    * GetBeamProfile *
                    ******************
}

procedure SetUpBeamProfile(var BeamProfileVars:
  BeamProfileVarType;
  Sensor: Integer;X,Y: Real;_ScanStep,_BkgndThreshold:
    Integer);
begin
  with BeamProfileVars[Sensor] do begin
    if X                >0 then Xpos           :=
            trunc(X);
    if Y                >= 0 then Ypos          :=
            trunc(Y);
    if _ScanStep        >= 0 then ScanStep      :=
            _ScanStep;
    if _BkgndThreshold  >= 0 then BkgndThreshold
             := _BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;

procedure GetBeamProfile(var BeamProfileVars: Beam
  ProfileVarType;
  Sensor: Integer);
var
  Pass,row,col,ADC,delay,waste: Integer;
  Xcorner,Ycorner: LongInt;
begin
```

```
with BeamProfileVars[Sensor] do begin
  XYweight        := 0;
  Xweight         := 0;
  Yweight         := 0;
  TotalWeight     := 0;  { for power, not removing
    noise till end}
  SaturationCount := 0;
  HiPoints        := 0;
  BeamFound       := false;
  for col:=1 to ColMax do ColWeight[col] :=0;
  for row:=1 to RowMax do RowWeight[row] := 0;
  Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
  Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
  MoveBeam(Xcorner,Ycorner);
  waste := 0;
  while waste < InitialDelay do inc(waste);
  PeakReading := 0;
  for col:=1 to ColMax do begin { prefer to return
    around edges after line }
    Xpos := Xcorner + col * ScanStep;
    Ypos := Ycorner;
    MoveBeam(Xpos,Ypos);
    waste:= 0;
    while waste < ColToColDelay do inc(waste);
    for row:=1 to RowMax do begin
      inc(Ypos,ScanStep);
      MoveInYAxis(Ypos);
      waste := 0;
      while waste < RowToRowDelay do inc(waste);
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
      if ADC < BkgndThreshold then ADC := 0
        else begin
          inc(HiPoints);
          BeamFound := true;
        end;
      if ADC > 210 then inc(SaturationCount);
      inc(TotalWeight,ADC);
```

```
          inc(ColWeight[col],ADC);
          inc(RowWeight[row],ADC);
          BeamMap[row,col] := ADC;
          inc(XYweight,ADC);
          inc(Xweight,Xpos * ADC);   { want floating
                                       point numbers here }
          inc(Yweight,Ypos * ADC);
        end; {for row}
      end; {for col}
      MaxSensorReading := PeakReading;
    end;
  end;

  {
                        ************
                        * GoodRead *
                        ************
  }

  procedure GoodRead(Sensor,iter: Integer);
  var
    Try: Integer;
  begin
    with BeamProfileVariables[Sensor] do begin
      Xaverage  := 0;
      Yaverage  := 0;
      TotWtAvg  := 0;
      GoodCount := 0;
      for Try := 1 to iter do begin
        SetupBeamProfile(BeamProfileVariables,
          Sensor,Xsp[Sensor],
          Ysp[Sensor],-1(ScanStep),-1{BkgndThreshold});
        GetBeamProfile(BeamProfileVariables,Sensor);
        if XYweight > 0 then begin
          Xaverage := Xweight / XYweight + Xaverage;
          Yaverage := Yweight / XYweight + Yaverage;
          TotWtAvg := TotWtAvg + TotalWeight - Sensor
```

```
            Bkgnd[Sensor];
          inc(GoodCount);
        end;
      end; {for Try}
      if GoodCount > 0 then begin
        Xaverage  := Xaverage / GoodCount;
        Yaverage  := Yaverage / GoodCount;
        TotWtAvg  := TotWtAvg / GoodCount;
      end;
      Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
    end;
  end;

{-------------------------------------------------------
                      HARDWARE ROUTINES
-------------------------------------------------------}
                          ***********
                          * IOSetUp *
                          ***********

}
procedure IOSetup;
begin
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Xpos := YRest;
  MoveBeam(Xpos,Ypos);
end;

{
                        ************
                        * FindBeam *
                        ************
}

procedure SetupFindBeam(var FindBeamVars: FindBeamVarType;
  Sensor: Integer;X,Y: Real;_SpiralSideMax,_SpiralTries,
```

```
  _ScanStep,_BkgndThreshold,_SpiralDelay,_NumCenterPasses:
  Integer);
begin
  with FindBeamVars[Sensor] do begin
    if X                >= 0 then Xval          :=
                           trunc(X);
    if Y                >= 0 then Xval          :=
                             trunc(Y);
    if _SpiralSideMax   >= 0 then SpiralSideMax
                             := _SpiralSideMax;
    if _SpiralTries     >= 0 then SpiralTries
                             := _SpiralTries;
    if _ScanStep        >= 0 then ScanStep      :       =
                             _ScanStep;
    if _SpiralDelay     >= 0 then SpiralDelay
                             := _SpiralDelay;
    if _BkgndThreshold  >= 0 then BkgndThreshold
                             := _BkgndThreshold;
    if _NumCenterPasses >=0 then NumCenterPasses
                             := _NumCenterPasses;
  end;
end;

function SpiralSearchForHold(var FindBeamVars:
  FindBeamVarType;
  Sensor: Integer): Boolean;
var
  i,ADC,SpiralSide,waste: Integer;
  X,Y: LongInt;
begin
  with FindBeamVars[Sensor] do begin
    X := trunc(Xval);
    Y := trunc(Yval);
    MoveBeam(X,Y);
    delay(SpiralDelay);
    BeamFound := false;
    SpiralSide := 1;
```

```
    while (SpiralSide <= SpiralSideMax) and not
      BeamFound do begin
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInAxis(X);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(X,ScanStep);
        inc(i);
      end;
      i :=1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInAxis(Y);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(Y,ScanStep);
        inc(i);
      end;
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := BeamFound;
  end;
end;

function FindBeam(var FindBeamVars:
  FindBeamVarType;Sensor: Integer;
  FindMethod: FindMethodType): Boolean;
```

```
const
  HoleSeparation = 600;
var
  i,ADC,pass,CantFind,SpiralSide: Integer;
  XspAv,YspAv: Real;
  X,Y,GoodX,GoodY: LongInt;
  BeamFound: Boolean;
begin
  FindBeam := true;
  with FindBeamVars[Sensor] do begin
    CantFind := 0;
    repeat
    X := trunc(Xval);
    Y := trunc(Yval);
    MoveBeam(X,Y); { gets us to best starting position }
{
  now spiral to search for holes
}
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,
        -1{SpiralSideMax},-1{SpiralTries},
        -1{ScanStep},-1{BkgndThreshold},
        -1{SpiralDelay},-1{NumCenterPasses});
      BeamFound := SpiralSearchForHole
        (FindBeamVariables,Sensor);
      if not BeamFound then begin
        SetupFindBeam(FindBeamVariables,Sensor,
          X,Y,SpiralSideMax * 2,
          -1{SpiralTries},-1{ScanStep},BkgndThreshold -1,
          -1{SpiralDelay},-1{NumCenterPasses});
        inc(CantFind);
      end;
    until (CantFind >= SpiralTries) or BeamFound;
    if CantFind >= SpiralTries then begin
      MoveBeam(Xval,Yval);
      FindBeam := false;
      exit;
    end;
```

```
  X := Xval;
  Y := Yval;

{
 Verify that we have 2 mil hole...

 1/2 mil   1 mil ) left rear 2 mil  4 mil  ) right front
                 ) of vat                   ) of vat
  4 mil    2 mil )           1 mil  1/2 mil )
       Sensor #1             Sensor #2

 We have the position of one of the four holes: 1/2, 1,
 2, or 4 mils.  We have to find the 2 mil hole closest to
 the vat.  Note that the orientation of the holes is
 different for Sensors 1 and 2.

 Hole Size Check Procedure:

 1.  We may have the position for the 1/2 or 1 mil hole.
     Move towards the vat in the X-axis and search for
     the 1 or 2 mil hole.  If the beam is seen, adopt the
     new coordinates.

 2.  Now we may have the position for the 4 mil hole.
     Move towards the vat in the Y-axis and search for
     the 2 mil hole.  If the beam is seen, use the new
     coordinates.

 if (Sensor < 3) and <FindMethod = Verify2MilHole) then
   begin
   GoodX := X;
   GoodY := Y;
   if X < 32767 then inc(X,HoleSeparation)
     else dec(X,HoleSeparation);
   SetupFindBeam(FindBeamVariables,Sensor,X,Y,
     SpiralSideMax div 2,-{SpiralTries},-1{ScanStep},
     -1{BkgndThreshold},-1{SpiralDelay},
```

```
        -1{NumberCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
        then begin
          GoodX := Xval;
          GoodY := Yval;
        end;
      X := GoodX;
      Y := GoodY;
      if Y < 32767 then inc(Y,HoleSeparation)
        else dec(Y,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,
        -1{SpiralSideMax},-1{Spiraltries},-1{ScanStep},
        -1{BkgndThreshold},-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
        then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
      X := GoodX;
      Y := GoodY;
    end;

  XspAv := 0;
  YspAv := 0;
  if BeamFound then begin
    for pass:=1 to NumCenterPasses do with
      BeamProfileVariables[Sensor] do begin
      CantFind := 0;
      { discard the first }
      SetupBeamProfile(BeamProfileVariables,Sensor,
        X,Y,4{ScanStep},trunc(Bk[Sensor] + 0.5));
      repeat
        GetBeamProfile(BeamProfileVariables,Sensor);
              if XYweight = 0 then inc(CantFind);
          until (XY weight > 0) or (Cantfind > 3);
          if CantFind > 3 then begin
```

```
            FindBeam := false;
            exit;
          end;
{
  Later select sensor hole size automatically
}
        X := trunc(Xweight / XYweight);
            { better coordinates }
        Y := trunc(Yweight / XYweight);
        Power[Sensor] := (TotalWeight -
          SensorBkgnd[Sensor])
          * (SensorCalib[Sensor,3]);
        XspAv := XspAv + Xweight / XYweight;
        YspAv := YspAv + Yweight / XYweight;
      end; {for pass}
      if NumCenterPasses > 1 then begin
        XspAv := XspAv / NumCenterPasses;
        YspAv := YspAv / NumCenterPasses;
      end;
    end; {ifBeamFound}
    Xval := X;      { update integer coordinates }
    Yval := Y;
    Xavg := XspAv   { update average coordinates }
    Yavg := YspAv
  end;
end;

function CoarseFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor], 50{SpiralSideMax},3{SpiralTries},
    16{ScanStep},trunc(Bk[Sensor] + 0.5)
    {BkgndThreshold},1{SpiralDelay},
    1{NumCenterPasses});
  if FindBeam(FindBeamVariables[Sensor],NormalFind)
    then begin
```

```
    Xsp[Sensor] := FindBeamVariables[Sensor],Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor],Yval;
  end;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
    Ysp[Sensor],50{SpiralSideMax},3{SpiralTries},
    16{ScanStep},trunc(Bk[Sensor] + 0.5)
    {BkgndThreshold},1{SpiralDelay},
    2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind)
    then begin
    Xsp[Sensor] := FindBeamVariables[Sensor]Xval;
    Xsp[Sensor] := FindBeamVariables[Sensor]Yval;
  end;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,
    Xsp[Sensor],Ysp[Sensor],50{SpiralSideMax},
    3{SpiralTries},16{ScanStep},trunc(Bk[Sensor]
    + 0.5){BkgndThreshold},100{SpiralDelay},
    2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,Verify2MilHole)
    then begin
    Xsp[Sensor] := FindBeamVariables[Sensor].
      Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].
      Yval;
  end;
  FineFindBeam := BeamFound;
end;
```

```
function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) then
    BeamFound := FineFindBeam(Sensor);
  FindLaserBeam := BeamFound;
end;

{ - - - - - }

function FindHoles2(Sensor: Integer): Boolean;
{
  When we want to use new starting point
    updates hole position
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;

function FindHoles(Sensor: Integer): Boolean;
{
  Updates hole position
  Needs to confirm that we have 2 mil hole
}
var
  BkRead: LongInt;
  ADC,row,col: Integer;
begin
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
  BkRead := 0;
  for col:=1 to ColMax do begin
    for row:=1 to Row Max do begin
      ADC := ReadLaserBoard(SensorChannel[Sensor];
      inc(BkRead,ADC);
    end;
  end;
```

```
    SensorBkgnd[Sensor] := BkRead;
    { remove noise for centering }
    Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax + 3;
    Xsp[Sensor] := XSensorPos[Sensor];  { will be
                              best starting point }
    Ysp[Sensor] := YSensorPos[Sensor];
    FindHoles := FindHoles2(Sensor);
  end;

  {-------------------------------------------------------}

  {
    Read BEAM.PRM parameters file
  }
  procedure ReadBeamParamsFile;
  var
    ParamFile: text;
    i,count,code,code1,code2,Sensor,Pinhole: Integer;
    cmd,ParamLine: String;

    procedure GetParamFileLine;
    begin
      if eof(ParamFile) then ParamLine := ''
      else repeat
        readln(ParamFile,ParamLine);
      until eof(ParamFile) or ((ParamLine <> '')
        and (ParamLine[1] <> '!'));
    end;

  begin
    {$I-}
    assign(ParamFile,ParamFileName);
    reset(ParamFile);
    if IOresult<>0 then begin
      assign(ParamFile,HomeDirectory+ParamFileName);
      reset(ParamFile);
      if IOresult<>0 then halt(3);
```

```
end;
{$I+}
GetParamFileLine;
if copy(ParamLine,1,1) = chr(34) then begin
        { handle quoted options line }
  i := pos(chr(34),copy(ParamLine,2,255));
  if i>0 then ParamLine[0] := chr(i);
  ParamLine := NoQuotes(ParamLine);
  GetParamFileLine;
end;
XRest := WordVal(ParamLine,code);
YRest := WordVal(copy(ParamLine,code +1, 255), code);
GetParamFileLine;
SensorNum := WordVal(ParamLine,code);
for Sensor:=1 to SensorNum do begin
  GetparamFileLine;
  Sensorchannel[Sensor] := WordVal(ParamLine,code);
  GetParamFileLine;
  XSensorPos[Sensor] := WordVal(ParamLine,code);
  YSensorPos[Sensor] := WordVal(copy(ParamLine,
    code+1,255),code);
  for Pinhole:=1 to 4 do begin GetParamFileLine;
    SensorCalib[Sensor,Pinhole] :=
      RealVal(ParamLine,code);
  end;
end; {for Sensor}
GetParamFileLine;
TempChannel := WordVal(ParamLine,code);
              { assume only one thermometer at first }
GetParamFileLine;
TempCalPts := WordVal(ParamLine,code;
for i:=1 to TempCalPts do begin
  { Resistance[i],Temperature[i] may not be integers }
  GetParamFileLine;
  Resistance[i]  := RealVal(ParamLine,code);
  Temperature[i] := RealVal(copy(ParamLine,
    code+1,255),code) + 273.15;
```

```
  end;
  for i:=1 to 3 do begin  { read in Fast, Medium,
                    & Slow Calib Delay values }
    Get ParamFileLine;
    CalibDelays[i,1] := LongVal(ParamLine,code1);
    CalibDelays[i,2] := trunc(RealVal(copy(ParamLine,
      code1+1,255),code2)*10+0.5)
    CalibDelays[i,3] := trunc(RealVal(copy(ParamLine,
      code2+code1+1,255),code)*10
  end;
  close(ParamFile);
  Resistance [0] := Resistance [1] * 2 - Resistance [2];
  Temperature[0] := Temperature[1] * 2 - Temperature[2];
end;

{-------------------------------------------------------}

procedure SetLaser;
begin
    IOSetup;
    for i:=1 to 3 do begin
      Xsp[i] := XSensorPos[i];
      Ysp[i] := YSensorPos[i];
  end;
  Xpos := XRest;   { Move beam to rest position }
  Ypos := YRest
  MoveBeam(Xpos,Ypos);
  delay(500);
end;

procedure MonitorLaserBeamPower;
begin
  println('Monitoring Laser Beam Power...');
  SetLaser;
  assign(PowerHistoryFile,PowerHistoryFileName);
  {$I-}
  append(PowerHistoryFile);
```

```
  if IOresult<> 0 then rewrite(PowerHistoryFile);
  ($I+}
  if IOresult = 0 then begin
    for Sensor := 1 to NumSensors do begin
      if not FindHoles(Sensor) then begin
        beep;
        beep;
        Power[Sensor] := 0;
      end else GoodRead(Sensor,10);
    end;
    writeln(PowerHistoryFile,TimeStamp + ' Sensor #1=',
      RealStr(Power[1],2), Sensor #2=',
      RealStr(Power[2]2));
    close(PowerHistoryFile);
  end else beep;
end;

procedure MonitorLaserBeamProfile;
var
  row,col: Integer;
begin
  println('Monitoring Laser Beam Profile...');
  SetLaser;
  assign(ProfileHistoryFile,ProfileHistoryFileName);
  {$I-}
  append (Profile History File);
  if IO result <>O then rewrite (Profile History File);
  {$I+}
  if IOresult = 0 then begin
    for Sensor := 1 to NumSensors do begin
      if FindHoles(Sensor) then begin
        writeln(ProfileHistoryFile,TimeStamp,'
          Sensor #',IntStr(Sensor),':');
        for col:=1 to ColMax do begin
          write(ProfileHistoryFile,' ');
          for row:=1 to RowMax do begin
            write(ProfileHistoryFile,BeamProfileVariables
```

```
                [Sensor],BeamMap[row,col];
              end; {for row}
              writeln(ProfileHistoryFile);
            end; {for col}
          end else begin
            beep;
            beep;
            writeln(ProfileHistory,file,TimeStamp,'
              Sensor #',InStr(Sensor),' beam not found');
        end;
      end;
      close(ProfileHistoryFile);
    end else beep;
end;

procedure ShowTitle;
begin
  TextMode(CBO);
  SingleBar := many(#196,45);
  writeln(SingleBar);
  writeln('   MONITOR, Laser Beam Monitor Program');
  writeln('   Version '+VersionId);
  writeln('   Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography System');
  writeln(SingleBar);
end;

{ - - M A I N   P R O G R A M - - - - }

begin { Monitor }
  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);
        { clear variables }
  OldExitProc := ExitProc;
  ExitProc := @ExitMonitor;
  CheckBreak := true;
  OutFileName := '';
  ReadBeamParamsFile;
```

```
  ReadCommandLine(CmdLine);
  CmdLine := upper(CmdLine);
  i := pos ('/T',CmdLine);
  if i > 0 then begin
    Transparent := true;
    delete(CmdLine,i,2);
  end;
  if not Transparent then begin
    ShowTitle;
    writeln;
    writeln;
  end;
  if pos('POWER',CmdLine) > 0 then MonitorLaserBeamPower;
  if pos('PROFILE',CmdLine) > 0 then
MonitorLaserBeamProfile;
  QuietSystemError('Success.');
end.
```

```
/*******************************************************/
/*              3D Systems Slice Program               */
/*         Copyright (C) 1988 3D Systems Inc.          */
/*                                                     */
/* PFP.H                                               */
/*                                                     */
/*******************************************************/

typedef unsigned long       UINT32;
typedef long                INT32;
typedef unsigned short      UINT16;
typedef short               INT16;
typedef unsigned char       BOOLEAN;
typedef unsigned char       UINT8;
typedef char                STRING;

#define SYS_SUCCESS         0
#define SYS_FAILURE         1

#define DISPLAY_OFF         0
#define DISPLAY_ON          1

#define LB_CODE             1
#define LH_CODE             2
#define NFUB_CODE           3
#define NFUH_CODE           4
#define NFUF_CODE           5
#define NFDB_CODE           6
#define NFDH_CODE           7
#define NFDF_CODE           8
#define FDB_CODE            9
#define FDH_CODE            10
#define FDF_CODE            11
#define FUB_CODE            12
#define FUH_CODE            13
#define FUF_CODE            14
```

```
#define TRUE                1
#define FALSE               0

#define MAX_POINTER_INDEX   16383

#define MAX_Z_VAL           65535
#define MAX_Z_INDEX         16383
#define MAX_X_VAL           65535
#define MAX_Y_VAL           65535

#define MAX_DBL             1E+38
#define MAX_UINT            65535
#define ZERO_DEF            1E-5
#define RADIANS_PER_DEGREE  0.01745329266666666666
#define MAX_HASH_INDEX      1501
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPCHL.H                                          */
/* PFP Cross-Hatch Line Header                       */
/*                                                   */
/*****************************************************/

struct hatch_point {
    UINT16              x_val;
    UINT16              y_val;
    UINT16              angle;
    UINT8               enter_flag;
    UINT16              triangle_number;
    struct hatch_point *next_point;
   };
typedef struct hatch_point HATCH_POINT_STRUC;

struct min_max_type {
  UINT16       min_x;
  UINT16       max_x;
  UINT16       min_y;
  UINT16       max_y;
  };

typedef struct min_max_type MIN_MAX_STRUC;

#define NULL_HATCH_POINT (HATCH_POINT_STRUC *) NULL
```

```
/*******************************************************/
/*              3D Systems Slice Program               */
/*        Copyright (C) 1988 3D Systems Inc.           */
/*                                                     */
/*                                                     */
/* PFPCURE.H                                           */
/*                                                     */
/*******************************************************/

struct cure_hash {
  UINT16          x;
  UINT16          y;
  OUT_SEG_STRUC   *seg_ptr;
  struct cure_hash *last_hash;
  struct cure_hash *next_hash;
  };

typedef struct cure_hash CURE_HASH_STRUC;

#define NULL_CURE_HASH   (CURE_HASH_STRUC *) NULL
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPHI.H                                           */
/* PFP Human Interface heading file                  */
/*                                                   */
/*****************************************************/


#define FILE_NAME_LENGTH 100

typedef struct
   {
    float    res_value;
    UINT16   y_flat_fill_spacing;
    UINT16   x_flat_fill_spacing;
    UINT16   y_hatch_spacing;
    UINT16   x_hatch_spacing;
    UINT16   angle_hatch_spacing;
    UINT16   z_spacing;
    UINT16   min_surface_angle;
    UINT16   min_intersect_angle;
    UINT16   axis;
    STRING   z_space_filename [FILE_NAME_LENGTH];
    STRING   internal_filename[FILE_NAME_LENGTH];
    STRING   segfile_name     [FILE_NAME_LENGTH];
    BOOLEAN  binary_input;
    BOOLEAN  debug_mode;
    BOOLEAN  display_on;
    BOOLEAN  centering_off;
    BOOLEAN  section_codes_off;
    BOOLEAN  attrib;
    BOOLEAN  in_mem;

    INT 17    cure;
```

```
} USER_INFO_STRUC;


#define RES_OPTION                    "-RES"
#define SEGFILE_OPTION                "-SF"
#define HATCH_FILE_OPTION             "-HF"
#define Y_HATCH_SPACE_OPTION          "-HY"
#define X_HATCH_SPACE_OPTION          "-HX"
#define ANGLE_HATCH_SPACE_OPTION      "-HA"
#define YF_HATCH_SPACE_OPTION         "-HFY"
#define XF_HATCH_SPACE_OPTION         "-HFX"
#define HELP_OPTION                   "-H"
#define DISPLAY_OPTION                "-D"
#define Z_SPACE_OPTION                "-ZS"
#define Z_SPACE_FILE_OPTION           "-ZSF"
#define CENTER_OPTION                 "-C"
#define SECTION_CODES_OFF_OPTION      "-NSC"
#define MIN_SURFACE_ANGLE_OPTION      "-MSA"
#define MIN_INTERSECT_ANGLE_OPTION    "-MIA"
#define X_AXIS_OPTION                 "-X"
#define Y_AXIS_OPTION                 "-Y"
#define BINARY_INPUT_OPTION           "-B"
#define DEBUG_OPTION                  "-DEBUG"
#define DISK_OPTION                   "-DISK"
#define ARIES_OPTION                  "-ARIES"
#define INTERNAL_OPTION               "-INTERNAL"
#define CURE_OPTION                    "-CURE"


#define RES_OPTION_CODE                   1
#define SEGFILE_OPTION_CODE               2
#define Y_HATCH_SPACE_OPTION_CODE         3
#define X_HATCH_SPACE_OPTION_CODE         4
#define ANGLE_HATCH_SPACE_OPTION_CODE     5
#define YF_HATCH_SPACE_OPTION_CODE        6
#define XF_HATCH_SPACE_OPTION_CODE        7
#define HELP_OPTION_CODE                  8
```

```
#define DEBUG_OPTION_CODE                9
#define DISPLAY_OPTION_CODE              10
#define Z_SPACE_OPTION_CODE              11
#define Z_SPACE_FILE_OPTION_CODE         12
#define CENTER_OPTION_CODE               13
#define SECTION_CODES_OFF_OPTION_CODE    14
#define MIN_SURFACE_ANGLE_OPTION_CODE    15
#define MIN_INTERSECT_ANGLE_OPTION_CODE  16
#define X_AXIS_OPTION_CODE               17
#define Y_AXIS_OPTION_CODE               18
#define BINARY_INPUT_OPTION_CODE         19
#define DISK_OPTION_CODE                 20
#define ARIES_OPTION_CODE                21
#define INTERNAL_OPTION_CODE             22
#define CURE_OPTION_CODE                 23
#define UNKNOWN_OPTION_CODE              24

#define DEFAULT_RES_VALUE   1.0
#define DEFAULT_SEGFILE_NAME  ""

#define MAX_FLOAT_LENGTH 9
#define MAX_INT_LENGTH 5
#define MAX_ANGLE 90
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPIDB.C                                            */
/*                                                     */
/*******************************************************/
#include <stdio.h>
#include <math.h>
#include <malloc.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"

extern INPUT_TRI_STRUC *tri_bucket_array[];
extern OUT_SEG_STRUC *FD_seg_bucket_array[];
extern OUT_SEG_STRUC *FU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFD_seg_bucket_array[];
extern UINT16        z_control_table[];
extern UINT16        all_min_z, all_max_z,
all_min_z_index, all_max_z_index;

/*******************************************************/
   general procedure

   find_z_binarily - binary search of z_control_table
    for layer z

/*******************************************************/

UINT16 get_z_binarily(z)

   UINT16 z;
```

```
{
   UINT16 low = 0, mid, high = MAX_Z_INDEX-1;

   while (low <= high) {
      mid = (low + high) / 2;
      if (z < z_control_table [mid])
         high = mid - 1;
      else if (z > z_control_table [mid])
         low = mid + 1;
      else
         return(mid);
    }
    if (z_control_table [low] > z)
       return(low);
    else return(high);

}  /* get_z_binarily */

/*********************************************************/
   local procedure
   get_offsets
/*********************************************************/

static void get_offsets (min_x, min_y, min_z, max_x,
  max_y, max_z,
                          res_value, x_offset, y_offset,
  z_offset)
    float min_x, min_y, min_z, max_x, max_y, max_z,
   res_value;
    float *x_offset, *y_offset, *z_offset;

{
   BOOLEAN res_error1,res_error2;
   float   x_range, y_range, z_range;
   float   round_flt();
```

```
   x_range = (max_x - min_x) * res_value;
   y_range = (max_y - min_y) * res_value;
   z_range = (max_z - min_z) * res_value;

   res_error1 = ((x_range > MAX_X_VAL) || (y_range >
MAX_Y_VAL) ||
                  (z_range > MAX_Z_VAL));

   res_error2 = (z_range > z_control_table
[MAX_Z_INDEX-1]);

   if (res_error1 || res_error2) {
      if (res_error1) {
         fprintf (stderr, "\n error - resolution value
too large for this");
         fprintf (stderr, " part\n");
      }
      if (res_error2) {
         fprintf (stderr, "\n error - z-value would
require more than");
         fprintf (stderr, " %u layers.\n",MAX_Z_INDEX);
      }
      fprintf (stderr,
      " min x = %f max x = %f min y = %f max y = %f min
z = %f max z = %f\n",
        min_x, max_x, min_y, max_y, min_z, max_z);
      exit (SYS_FAILURE);
   }

   *x_offset = round_flt(((MAX_X_VAL - x_range) / 2.0) -
(min_x * res_value));
   *y_offset = round_flt(((MAX_Y_VAL - y_range) / 2.0) -
(min_y * res_value));
   *z_offset = round_flt(((z_control_table
[MAX_Z_INDEX-1] - z_range) / 2.0) - (min_z *
res_value));
```

```
} /* get_offsets */

/*********************************************************/
   local procedure
   check_tri_dimensions
/*********************************************************/

static void check_tri_dimensions (tri_ptr)
   INPUT_TRI_STRUC *tri_ptr;

{
   if ((tri_ptr-> x1 > MAX_X_VAL) || (tri_ptr->
      x1 < 0.0) ||
       (tri_ptr-> x2 > MAX_X_VAL) || (tri_ptr-> x2 <
         0.0) ||
       (tri_ptr-> x3 > MAX_X_VAL) || (tri_ptr-> x3 <
         0.0) ||
       (tri_ptr-> y1 > MAX_Y_VAL) || (tri_ptr-> y1 <
         0.0) ||
       (tri_ptr-> y2 > MAX_Y_VAL) || (tri_ptr-> y2 <
         0.0) ||
       (tri_ptr-> y3 > MAX_Y_VAL) || (tri_ptr-> y3 <
         0.0) ||
       (tri_ptr-> z1 > z_control_table [MAX_Z_INDEX-1])
          || (tri_ptr-> z1 < 0.0) ||
       (tri_ptr-> z2 > z_control_table [MAX_Z_INDEX-1])
          || (tri_ptr-> z2 < 0.0) ||
       (tri_ptr-> z3 > z_control_table [MAX_Z_INDEX-1])
         || (tri_ptr-> z3 < 0.0))
      {
       fprintf (stderr, "\n error - dimensions of
        uncentered object");
       fprintf (stderr, " exceed limits.  Check
        Resolution value. \n");
       exit (SYS_FAILURE);
      }
```

```
} /* check_tri_dimensions */

/*******************************************************/
   local procedure
   res_triangles
/*******************************************************/
static void res_triangles (tri_list, x_offset, y_offset,
z_offset,
                           res_value, centering_off)

   INPUT_TRI_STRUC *tri_list;
   float           x_offset, y_offset, z_offset,
                   res_value;
   BOOLEAN         centering_off;
{
   INPUT_TRI_STRUC *tri_ptr;
   float            round_flt();

   tri_ptr = tri_list;

   if (centering_off)
      while (tri_ptr != NULL_INPUT_TRI)
         {
          tri_ptr-> x1 = round_flt(tri_ptr-> x1 *
            res_value);
          tri_ptr-> x2 = round_flt(tri_ptr-> x2 *
            res_value);
          tri_ptr-> x3 = round_flt(tri_ptr-> x3 *
            res_value);

          tri_ptr-> y1 = round_flt(tri_ptr-> y1 *
            res_value);
          tri_ptr-> y2 = round_flt(tri_ptr-> y2 *
            res_value);
          tri_ptr-> y3 = round_flt(tri_ptr-> y3 *
            res_value);
```

```
            tri_ptr-> z1 = round_flt(tri_ptr-> z1 *
              res_value);
            tri_ptr-> z2 = round_flt(tri_ptr-> z2 *
              res_value);
            tri_ptr-> z3 = round_flt(tri_ptr-> z3 *
              res_value);

            check_tri_dimensions (tri_ptr);
            tri_ptr = tri_ptr-> next_input_tri;
           }
      else
         while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> x1 = round_flt((tri_ptr-> x1 *
              res_value) + x_offset);
            tri_ptr-> x2 = round_flt((tri_ptr-> x2 *
              res_value) + x_offset);
            tri_ptr-> x3 = round_flt((tri_ptr-> x3 *
              res_value) + x_offset);

            tri_ptr-> y1 = round_flt((tri_ptr-> y1 *
              res_value) + y_offset);
            tri_ptr-> y2 = round_flt((tri_ptr-> y2 *
              res_value) + y_offset);
            tri_ptr-> y3 = round_flt((tri_ptr-> y3 *
              res_value) + y_offset);

            tri_ptr-> z1 = round_flt((tri_ptr-> z1 *
              res_value) + z_offset);
            tri_ptr-> z2 = round_flt((tri_ptr-> z2 *
              res_value) + z_offset);
            tri_ptr-> z3 = round_flt((tri_ptr-> z3 *
              res_value) + z_offset);
            tri_ptr = tri_ptr-> next_input_tri;
           }

      /* res_triangles */
```

```
/*****************************************************/
   local procedure
   remove_collapsed_triangles
/*****************************************************/

static void remove_collapsed_triangles (tri_list)

   INPUT_TRI_STRUC **tri_list;
{
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
   INPUT_TRI_STRUC head_rec;
   float           Ax,Ay,Az;
   float           Bx,By,Bz;
   float           Cx,Cy,Cz;
   unsigned long   point_count = 0,line_count = 0;
   BOOLEAN         delete_tri;

   tri_ptr = *tri_list;
   head_rec .next_input_tri = tri_ptr;
   back_ptr = &head_rec;

   while (tri_ptr != NULL_INPUT_TRI) {
     delete_tri = FALSE;
     if ((tri_ptr->x1 == tri_ptr->x2) && (tri_ptr->x2 ==
      tri_ptr->x3) &&
         (tri_ptr->y1 == tri_ptr->y2) && (tri_ptr->y2 ==
           tri_ptr->y3) &&
         (tri_ptr->z1 == tri_ptr->z2) && (tri_ptr->z2 ==
           tri_ptr->z3)) {
       delete_tri = TRUE;
       point_count++;
       }
     else {
       Ax = tri_ptr-> x2 - tri_ptr-> x1;
       Ay = tri_ptr-> y2 - tri_ptr-> y1;
       Az = tri_ptr-> z2 - tri_ptr-> z1;
       Bx = tri_ptr-> x3 - tri_ptr-> x1;
```

```
        By = tri_ptr-> y3 - tri_ptr-> y1;
        Bz = tri_ptr-> z3 - tri_ptr-> z1;
        Cx = fabs(Ay*Bz-Az*By);
        Cy = fabs(Az*Bx-Ax*Bz);
        Cz = fabs(Ax*By-Ay*Bx);
        if ((Cx < ZERO_DEF) && (Cy < ZERO_DEF) &&
          (Cz < ZERO_DEF)) {
          delete_tri = TRUE;
          line_count++;
          };
        };
      if (delete_tri) {
        back_ptr-> next_input_tri = tri_ptr->
next_input_tri;
        mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
        tri_ptr = back_ptr-> next_input_tri;
        }
      else {
        tri_ptr = tri_ptr-> next_input_tri;
        back_ptr = back_ptr-> next_input_tri;
        };
      };

   *tri_list = head_rec .next_input_tri;

   if (point_count > 0)
      fprintf (stderr, "\n warning - %lu collapsed point
        triangles removed \n", point_count);
   if (line_count > 0)
      fprintf (stderr, "\n warning - %lu collapsed line
        triangles removed \n", line_count);

} /* remove_collapsed_triangles */
```

```
/*******************************************************/
   local procedure
   get_z_output_coord
/*******************************************************/

static UINT16 get_z_output_coord (z)

   UINT16 z;
{
   UINT16 i, greater_z, lesser_z;

   i = get_z_binarily(z);
   greater_z = z_control_table [i];
   if (z == z_control_table [i])
      lesser_z = z_control_table [i];
   else
      lesser_z = z_control_table [i-1];
/* printf("get_z_output_coord:  z=%u i=%u lesser_z=%u
greater_z=%u\n",z,i,lesser_z,greater_z); */

   if ((lesser_z == 0) || (((greater_z - z) <= (z -
lesser_z)) &&
                                   (greater_z !=
z_control_table [MAX_Z_INDEX-1])))
      return (greater_z);
   else
      return (lesser_z);

} /* get_z_output_coord */

/*******************************************************/
   local procedure
   adjust_z_coords
/*******************************************************/

static void adjust_z_coords (tri_list, z_spacing)
```

```
    INPUT_TRI_STRUC *tri_list;
    UINT16          z_spacing;

{
    INPUT_TRI_STRUC *tri_ptr;
    float           round_flt();
    UINT16          get_z_coord();

    if (z_spacing != 1)
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = round_flt((float) tri_ptr-> z1
             / z_spacing) * z_spacing;
            tri_ptr-> z2 = round_flt((float) tri_ptr-> z2
             / z_spacing) * z_spacing;
            tri_ptr-> z3 = round_flt((float) tri_ptr-> z3
             / z_spacing) * z_spacing;
            tri_ptr = tri_ptr-> next_input_tri;
           }
       }
    else
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = get_z_output_coord ((UINT16)
             tri_ptr-> z1);
            tri_ptr-> z2 = get_z_output_coord ((UINT16)
             tri_ptr-> z2);
            tri_ptr-> z3 = get_z_output_coord ((UINT16)
             tri_ptr-> z3);
            tri_ptr = tri_ptr-> next_input_tri;
           }
       }
```

```
} /* adjust_z_coords */

/*******************************************************/
   local procedure
   output_internal_triangles
/*******************************************************/

static void output_internal_triangles
(user_info,tri_list)

USER_INFO_STRUC *user_info;
INPUT_TRI_STRUC **tri_list;

{
FILE *InternalFile;
INPUT_TRI_STRUC *tri_ptr;
UINT32 count,tri_count;
float zero,i,j,k;

   zero = 0;
   tri_ptr = *tri_list;
   tri_count = 0;
   while (tri_ptr != NULL_INPUT_TRI) {
      tri_count++;
      tri_ptr = tri_ptr-> next_input_tri;
   }
   if (tri_count>0) {
      fprintf(stderr, " writing internal triangles
       output file ...\n");
      InternalFile = fopen(user_info->
        internal_filename,"wb");

      fprintf(InternalFile,"Slice Internal Triangle
        File");
      count = 28;
      while (count<80) {              /* should write
        out copy of org header */
```

```
            fprintf(InternalFile," ");
            count++;
         }
         fwrite(&tri_count,4,1,InternalFile);  /* write out
          # of triangle facets */ tri_ptr = *tri_list;
         while (tri_ptr != NULL_INPUT_TRI) {
            count = 3;
            if (tri_ptr->normal_info&NORMAL_X_POSITIVE)
             i = 1;
            else if
(tri_ptr->normal_info&NORMAL_X_NEGATIVE) i = -1;
                  else {
                    i = 0;
                    count--;
                    };
            if (tri_ptr->normal_info&NORMAL_Y_POSITIVE)
             j = 1;
            else if
(tri_ptr->normal_info&NORMAL_Y_NEGATIVE) j = -1;
                  else {
                    j = 0;
                    count--;
                    };
            if (tri_ptr->normal_info&NORMAL_Z_POSITIVE)
               k = 1;
            else if
(tri_ptr->normal_info&NORMAL_Z_NEGATIVE) k = -1;
                  else {
                    k = 0;
                    count--;
                    };
            switch (count) {
              case 2 : i *= 0.707106781;
                       j *= 0.707106781;
                       k *= 0.707106781;
                       break;
              case 3 : i *= 0.577350269;
```

```
                        j *= 0.577350269;
                        k *= 0.577350269;
                        break;
                  };
               fwrite(&i,4,1,InternalFile);
               fwrite(&j,4,1,InternalFile);
               fwrite(&k,4,1,InternalFile);

               fwrite(tri_ptr,36,1,InternalFile);   /* write
out triangle vertices */
               fwrite(&zero,2,1,InternalFile);            /*
attribute data (none) */
               tri_ptr = tri_ptr-> next_input_tri;
            }
            fclose(InternalFile);
         }
      }
      /*********************************************************/
         local function
         flat_tri
      /*********************************************************/

      static flat_tri (tri_ptr)

         INPUT_TRI_STRUC *tri_ptr;

      {
         return
           ((tri_ptr-> z1 == tri_ptr-> z2) && (tri_ptr-> z2 ==
            tri_ptr-> z3));

      } /* flat_tri */

      /*********************************************************/
         local function
         near_flat_tri
      /*********************************************************/
```

```
static near_flat_tri (tri_ptr, min_angle)

   INPUT_TRI_STRUC *tri_ptr;
   INT16           min_angle;
{
   double dx1, dy1, dz1, dx2, dy2, dz2, dx, dy, dz, d,
    angle;
   if (!min_angle)
      return (FALSE);
   else
      {
       dx1 = tri_ptr-> x1 - tri_ptr-> x2;
       dy1 = tri_ptr-> y1 - tri_ptr-> y2;
       dz1 = tri_ptr-> z1 - tri_ptr-> z2;

       dx2 = tri_ptr-> x1 - tri_ptr-> x3;
       dy2 = tri_ptr-> y1 - tri_ptr-> y3;
       dz2 = tri_ptr-> z1 - tri_ptr-> z3;

       dx = dy1 * dz2 - dz1 * dy2;
       dy = dz1 * dx2 - dx1 * dz2;
       dz = dx1 * dy2 - dy1 * dx2;

       d = sqrt (dx * dx + dy * dy);

       if (d == 0) angle = 0;
       else angle = 90.0 - fabs (atan (dz / d) /
RADIANS_PER_DEGREE);

       return (angle < (double) min_angle);
      }

} /* near_flat_tri */
/*************************************************
local function
near_flat_tir
*************************************************/
```

```
static near_flat_tri (tri_ptr, min_angle)
  INPUT_TRI_STRUC *tri_ptr;
  INT16           min_angle;
{
  double dx1, dy1, dz1, dx2, dy2, dz2, dx, dy, dz,
    d, angle;
  if (!min_angle)
      return (FALSE);
  else
    {
      dx1 = tri_ptr-> x1 - tri_ptr-> x2;
      dy1 = tri_ptr-> y1 - tri-ptr-> y2;
      dz1 = tri_ptr-> z1 - tri_ptr-> z2;

      dx = dy1 * dz2 - dz1 * dy2;
      dy = dz1 * dx2 - dx1 * dz2;
      dz = dx1 * dy2 - dy1 * dx2;

      d = sqrt (dx * dx + dy * dy);

      if (d == 0 angle = 0;
      else angle = 90.0 - fabs (atan (dz / d) /
        RADIANS_PER_DEGREE);
      return (angle < (double) min_angle);
    }
} /* near_flat_tri */
/*******************************************************
cross_product
Computes the cross product of 2 vectors
*******************************************************
void cross_product(a1,a2,a3,b1,b2,b3,i,j,k)
  double a1,a2,a3; /* Point A */
  double b1,b2,b3; /* Point B */
  double *i,*j,*k; /* A X B*/
{
/*
  Normal i,j,k - AXB = (a2*b3 - a3*b2)i + (a3*b1 -
```

```
    a1*b3)j + (a1*b2 - a2*b1)k
*/
  Normal i,j,k - AxB = (a2*b3 - a3*b2)i + (a3*b1 -
    a1*b3)j + (a1*b2 - a2*b1)k
*/
*i = a2*b3 - a3*b2;
*j = a3*b1 - a1*b3;
*k = a1*b2 - a2*b1;
};
/*********************************************************/
   recompute_normals

   Recomputes the normals of the triangles after they
    have been deformed by adjusting the Z coords.

   Version 252 - Only recompute X and Y

/*********************************************************/
static recompute_normals(tri_list,axis)
  INPUT_TRI_STRUC *tri_list;
  INT16           axis;
  {
  UINT8  flag;
  double a1,a2,a3;
  double b1,b2,b3;
  double i,j,k;
  double temp;

  /*
    Vector A - Vertex 2 to Vertex 3 of input
    Vector B - Vertex 2 to Vertex 1 of input

    Note: Vectors change if axis is not inputs (-x -y).
Must adjust for
          these cases.
  */
  while (tri_list != NULL_INPUT_TRI) {
```

```
        switch (axis) {
          case 0 : a1 = tri_list->x3 - tri_list->x2;
            /* No axis swap */
                   a2 = tri_list->y3 - tri_list->y2;
                   a3 = tri_list->z3 - tri_list->z2;
                   b1 = tri_list->x1 - tri_list->x2;
                   b2 = tri_list->y1 - tri_list->y2;
                   b3 = tri_list->z1 - tri_list->z2;
                   break;
          case 1 : a3 = tri_list->x3 - tri_list->x2;
            /* X and Z axis swapped */
                   a2 = tri_list->y3 - tri_list->y2;
                   a1 = tri_list->z3 - tri_list->z2;
                   b3 = tri_list->x1 - tri_list->x2;
                   b2 = tri_list->y1 - tri_list->y2;
                   b1 = tri_list->z1 - tri_list->z2;
                   break;
          case 2 : a1 = tri_list->x3 - tri_list->x2;
            /* Y and Z axis swapped */
                   a3 = tri_list->y3 - tri_list->y2;
                   a2 = tri_list->z3 - tri_list->z2;
                   b1 = tri_list->x1 - tri_list->x2;
                   b3 = tri_list->y1 - tri_list->y2;
                   b2 = tri_list->z1 - tri_list->z2;
                   break;
          };
        cross-product (a1,a2,a3,b1,b2,b3,&i,&j,&k);
        switch (axis) {                          /* reswap
    flag info */
          case 1: temp = k;
                  k = i;
                  i = temp;
                  break;
          case 2: temp = k;
                  k = j;
                  j = temp;
                  break;
```

```
      };
    flag = tri_list->normal_info &
(NORMAL_Z_POSITIVE+NORMAL_Z_NEGATIVE);

    if (fabs(i) > ZERO_DEF) {
      if (i < 0) flag |= NORMAL_X_NEGATIVE;
      else flag |= NORMAL_X_POSITIVE;
      };

    if (fabs(j) > ZERO_DEF) {
      if (j < 0) flag |= NORMAL_Y_NEGATIVE;
      else flag |= NORMAL_Y_POSITIVE;
      };

/*
    if (fabs(k) > ZERO_DEF) {
      if (k < 0) flag |= NORMAL_Z_NEGATIVE;
      else flag |= NORMAL_Z_POSITIVE;
      };
*/

    tri_list->normal_info = flag;
    tri_list = tri_list->next_input_tri;
    };
  };

/*****************************************************/
   local procedure
/*****************************************************/

static void get_tri_lists (in_file, user_info,
res_value, min_angle,
                           centering_off, z_spacing,
                           scan_tri_list, flat_tri_list,
near_flat_tri_list)

   FILE            *in_file;
```

```
   USER_INFO_STRUC *user_info;
   float           res_value;
   INT16           min_angle;
   BOOLEAN         centering_off;
   UINT16          z_spacing;
   INPUT_TRI_STRUC **scan_tri_list;
   INPUT_TRI_STRUC **flat_tri_list;
   INPUT_TRI_STRUC **near_flat_tri_list;

{
   INPUT_TRI_STRUC *input_tri_list, *tri_ptr, *temp_ptr;
   unsigned long   up_scan_tri_count = 0,
dn_scan_tri_count = 0;
   unsigned long   up_flat_tri_count = 0,
dn_flat_tri_count = 0;
   unsigned long   up_near_flat_tri_count = 0,
dn_near_flat_tri_count = 0;
   float           min_x, min_y, min_z, max_x, max_y,
max_z;
   float           x_offset, y_offset, z_offset;
   long            resoluted_min_z,resoluted_max_z;

   min_x = MAX_X_VAL;
   min_y = MAX_Y_VAL;
   min_z = z_control_table [MAX_Z_INDEX-1];

   max_x = 0.0 - MAX_X_VAL;
   max_y = 0.0 - MAX_Y_VAL;
   max_z = 0.0 - z_control_table [MAX_Z_INDEX-1];

/*
printf ("\n get_tri_lists - begin \n");
*/
   fprintf (stderr, "\n loading input database ...\n");

   if (user_info-> binary_input)
      pfp_get_input_tri_list_bin (in_file,
```

```
       &input_tri_list, &min_x, &min_y,
                                    &min_z, &max_x,
&max_y, &max_z, user_info);
   else
      pfp_get_input_tri_list (in_file, &input_tri_list,
        &min_x, &min_y,
                                  &min_z, &max_x, &max_y,
&max_z,user_info);

   resoluted_min_z = round_flt(min_z * res_value);
   resoluted_max_z = round_flt(max_z * res_value);
   if (resoluted_min_z < z_control_table[0]) {
     if (*(user_info-> z_space_filename) != '\0')
       fprintf(stderr,"\nError - Part begins below
Variable Z Spacing Table\n");
     else
       fprintf(stderr,"\nError - Part begins below
minimum Z value\n");
     fprintf(stderr," Z Spacing Table: %u Part:
%ld\n",z_control_table[0],resoluted_min_z);
     exit (SYS_FAILURE);
     };
   if (resoluted_max_z > z_control_table[MAX_Z_INDEX-1])
{
     if (*(user_info-> z_space_filename) != '\0')
       fprintf(stderr,"\nError - Part extends past the
Variable Z Spacing Table of %u
layers\n",(MAX_Z_INDEX-1));
     else
       fprintf(stderr,"\nError - Part extends past the
maximum Z value of %u layers\n",(MAX_Z_INDEX-1));
     fprintf(stderr," Z Spacing Table: %u Part:
%u\n",z_control_table[MAX_Z_INDEX-1],resoluted_max_z);
     exit (SYS_FAILURE);
     };
   get_offsets (min_x, min_y, min_z, max_x, max_y,
max_z, res_value,
```

```
                &x_offset, &y_offset, &z_offset);
   res_triangles (input_tri_list, x_offset, y_offset,
z_offset,
                    res_value, centering_off);
   adjust_z_coords (input_tri_list, z_spacing);
   remove_collapsed_triangles (&input_tri_list);
   recompute_normals(input_tri_list,user_info->axis);
   if (*(user_info-> internal_filename) > '\0')
      output_internal_triangles
(user_info,&input_tri_list);
   while (input_tri_list != NULL_INPUT_TRI)
      {
       tri_ptr = input_tri_list;
       input_tri_list = input_tri_list-> next_input_tri;
       if (flat_tri (tri_ptr))
          {
           tri_ptr->normal_info = (tri_ptr->normal_info
              &(255-NORMAL_X_NEGATIVE-NORMAL_X_POSITIVE
              _NORMAL_Y_NEGATIVE-NORMAL_Y_POSITIVE))
                                  | FLAT_TRI;
           tri_ptr-> next_input_tri = *flat_tri_list;
           *flat_tri_list = tri_ptr;
           if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
              up_flat_tri_count++;
           else
              dn_flat_tri_count++;
          }
       else if (near_flat_tri (tri_ptr, min_angle))
          {
           temp_ptr = (INPUT_TRI_STRUC *) mem_alloc
(sizeof (INPUT_TRI_STRUC));
           copy (tri_ptr, temp_ptr, sizeof
(INPUT_TRI_STRUC));
           temp_ptr-> next_input_tri = *scan_tri_list;
           *scan_tri_list = temp_ptr;
```

```
            tri_ptr-> next_input_tri =
*near_flat_tri_list;
            *near_flat_tri_list = tri_ptr;
            if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
               up_near_flat_tri_count++;
            else
               dn_near_flat_tri_count++;
            }
         else
            {
            tri_ptr-> next_input_tri = *scan_tri_list;
            *scan_tri_list = tri_ptr;
            if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
               up_scan_tri_count++;
            else
               dn_scan_tri_count++;
            }
      }
   fprintf (stderr, "\n    %5lu scan triangles loaded
 (%5lu up facing, %5lu down facing)\n",
                        up_scan_tri_count +
dn_scan_tri_count,
                        up_scan_tri_count,
dn_scan_tri_count);
   fprintf (stderr, "    %5lu near flat triangles loaded
(%5lu up facing, %5lu down facing)\n",
                        up_near_flat_tri_count +
dn_near_flat_tri_count,
                        up_near_flat_tri_count,
dn_near_flat_tri_count);
   fprintf (stderr, "    %5lu flat triangles loaded
(%5lu up facing, %5lu down facing)\n\n",
                        up_flat_tri_count +
dn_flat_tri_count,
```

```
                        up_flat_tri_count,
dn_flat_tri_count);

/*
printf ("\n get_tri_lists - end \n");
*/

} /* get_tri_lists */

/*******************************************************/
   local procedure
   insert_tri
/*******************************************************/

static void insert_tri (input_tri, z_bucket_array)
   INPUT_TRI_STRUC *input_tri;
   INPUT_TRI_STRUC **z_bucket_array;
{
   float          min_z;
   UINT16         min_z_value;
   UINT16         i;

   pfp_get_min_z (input_tri, &min_z);
   min_z_value = (UINT16) min_z;
   i = get_z_binarily(min_z_value);

   input_tri-> next_input_tri = z_bucket_array [i];
   z_bucket_array [i] = input_tri;

} /* insert_tri */

void tri_bucket_sort (tri_list, a_tri_bucket_array)

   INPUT_TRI_STRUC **tri_list;
   INPUT_TRI_STRUC **a_tri_bucket_array;
```

```
{
   INPUT_TRI_STRUC *tri_ptr, **bucket_ptr;
   INT16           i;

/*
printf ("\n tri_bucket_sort - begin \n");
*/
   bucket_ptr = a_tri_bucket_array;

   for (i = 0; i < MAX_Z_INDEX; i++, bucket_ptr++)
      *bucket_ptr = NULL_INPUT_TRI;

   while (*tri_list != NULL_INPUT_TRI)
      {
       tri_ptr = (*tri_list)-> next_input_tri;
       insert_tri (*tri_list, a_tri_bucket_array);
       *tri_list = tri_ptr;
      }

   *tri_list = NULL_INPUT_TRI;

/*
printf ("\n tri_bucket_sort - end \n");
*/
} /* tri_bucket_sort */

void pfp_get_input_database (in_file, user_info)

   FILE            *in_file;
   USER_INFO_STRUC *user_info;
{
   INPUT_TRI_STRUC *scan_tri_list;
   INPUT_TRI_STRUC *flat_tri_list;
   INPUT_TRI_STRUC *near_flat_tri_list;
   UINT16          z;
```

```
   scan_tri_list      = NULL_INPUT_TRI;
   flat_tri_list      = NULL_INPUT_TRI;
   near_flat_tri_list = NULL_INPUT_TRI;

   get_tri_lists (in_file, user_info,
                  user_info-> res_value, user_info->
min_surface_angle,
                  user_info-> centering_off, user_info->
z_spacing,
                  &scan_tri_list, &flat_tri_list,
&near_flat_tri_list);

  for (z = 0; z < MAX_Z_INDEX; z++)
    {
     FU_seg_bucket_array[z] = NULL_SEG;
     FD_seg_bucket_array[z] = NULL_SEG;
     NFU_seg_bucket_array[z] = NULL_SEG;
     NFD_seg_bucket_array[z] = NULL_SEG;
     }

   if (scan_tri_list != NULL_INPUT_TRI)
      tri_bucket_sort (&scan_tri_list,
tri_bucket_array);

   if (flat_tri_list != NULL_INPUT_TRI)
      process_flat_tri_list (flat_tri_list);

   if (near_flat_tri_list != NULL_INPUT_TRI)
      process_near_flat_tri_list (near_flat_tri_list);

} /* pfp_get_input_database */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPRIE.C                                          */
/*                                                   */
/* PFP Remove Internal Edges Module                  */
/*****************************************************/

#include <stdio.h>
#include <math.h>
#include "pfp.h"
#include "pfpseg.h"
#include "pfputl.h"

static OUT_SEG_STRUC *hash_table     [MAX_HASH_INDEX];
static OUT_SEG_STRUC *end_hash_table [MAX_HASH_INDEX];
static BOOLEAN       hash_table_initialized = FALSE;

/*****************************************************/
   local procedure
   order_seg_points
/*****************************************************/

static void order_seg_points (seg_ptr)
  OUT_SEG_STRUC *seg_ptr;
  {
  UINT16        temp;

  if ((seg_ptr-> x_1 > seg_ptr-> x_2) || ((seg_ptr->x_1
== seg_ptr->x_2) &&
                                          (seg_ptr->y_1
>  seg_ptr->y_2))) {
    temp = seg_ptr-> x_1;
    seg_ptr-> x_1 = seg_ptr-> x_2;
    seg_ptr-> x_2 = temp;
    temp = seg_ptr-> y_1;
```

```
    seg_ptr-> y_1 = seg_ptr-> y_2;
    seg_ptr-> y_2 = temp;
    };

} /* order_seg_points */

/*******************************************************/
   local procedure
   insert_segment
/*******************************************************/

static void insert_segment (seg_ptr)
  OUT_SEG_STRUC  *seg_ptr;
  {
  OUT_SEG_STRUC  *scan_ptr,*back_ptr;
  double         hash_key;
  UINT16         hash_index;

  hash_key = (double) MAX_Y_VAL * seg_ptr-> x_1 +
seg_ptr-> y_1;
  hash_index = (UINT16) fmod(hash_key,(double)
MAX_HASH_INDEX);
  scan_ptr = hash_table [hash_index];
  while ((scan_ptr != NULL_SEG) &&
         !((scan_ptr-> x_1 == seg_ptr-> x_1) &&
           (scan_ptr-> y_1 == seg_ptr-> y_1) &&
           (scan_ptr-> x_2 == seg_ptr-> x_2) &&
           (scan_ptr-> y_2 == seg_ptr-> y_2))) {
    back_ptr = scan_ptr;
    scan_ptr = scan_ptr-> next_seg;
    };

  if (scan_ptr == NULL_SEG) {
    if (hash_table[hash_index] == NULL)
hash_table[hash_index] = seg_ptr;
    else end_hash_table[hash_index]->next_seg = seg_ptr;
    seg_ptr->next_seg = NULL_SEG;
```

```
    end_hash_table[hash_index] = seg_ptr;
    }
  else {
      if (hash_table[hash_index] == scan_ptr)
hash_table[hash_index] = scan_ptr->next_seg;
      else {
        back_ptr->next_seg = scan_ptr->next_seg;
        if (end_hash_table[hash_index] == scan_ptr)
end_hash_table[hash_index] = back_ptr;
        };
      mem_free (scan_ptr,sizeof(OUT_SEG_STRUC));
    mem_free (seg_ptr,sizeof(OUT_SEG_STRUC));
    };
  }; /* insert_segment */

/*******************************************************/
   exported procedure
   pfp_remove_duplicate_edges
/*******************************************************/

void pfp_remove_duplicate_edges (seg_list)
  OUT_SEG_STRUC **seg_list;
  {
  OUT_SEG_STRUC *seg_ptr;
  UINT16        i;

  if (!hash_table_initialized) {
    for (i = 0; i < MAX_HASH_INDEX; i++) hash_table [i]
= NULL_SEG;
    hash_table_initialized = TRUE;
    };

  while (*seg_list != NULL_SEG) {
    seg_ptr = *seg_list;
    *seg_list = seg_ptr->next_seg;
    if ((seg_ptr->x_1 == seg_ptr->x_2) &&
        (seg_ptr->y_1 == seg_ptr->y_2)) {
```

```
            mem_free(seg_ptr,sizeof(OUT_SEG_STRUC));
            }
          else {
            order_seg_points (seg_ptr);
            insert_segment (seg_ptr);
            };
          };

        for (i = 0; i < MAX_HASH_INDEX; i++)
          if (hash_table[i] != NULL_SEG) {
            end_hash_table[i]->next_seg = *seg_list;
            *seg_list = hash_table[i];
            hash_table [i] = NULL_SEG;
            };

      } /* pfp_remove_duplicate_edges */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPRIF.C                                          */
/*                                                   */
/* PFP Read Input File Module                        */
/*****************************************************/

#include <stdio.h>
#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"

#define NORMAL_FLAG_CHAR 'f'
#define NORMAL_MIN 1E-05

static UINT16       tri_count = 0;

/*****************************************************/
   local function
   is_legal_char
   Returns a value that indicates if the char parameter
   is within the legal character set.
/*****************************************************/

static BOOLEAN is_legal_char (ch)
   STRING ch;
{
   return ((ch >= ' ') && (ch <= '~'));
          /* the next line of code stops the '+' and '-'
   illegal char messages
             but causes other problems:
               run #1:  got memory parity error, system
                        halted
```

```
                    run #2:  got 'invalid input file format'
                             after several minutes and
                             program stopped, no other
                             information then commented out
                             line and recompiled to make it
                             like it was before  */
} /* is_legal_char */
/*******************************************************/
   local procedure
   get_line
   Reads characters from the input file into a line
      buffer up to the next newline char.  Checks for
      error.  Indicates if the file is eof.
/*******************************************************/

static void get_line (input_file, line_buf, error, eof)
   FILE    *input_file;
   STRING  *line_buf;
   BOOLEAN *error;
   BOOLEAN *eof;
{
   short status;
   BOOLEAN eoln;
   BOOLEAN char_error = FALSE;
   BOOLEAN leading_blanks_skipped = FALSE;

   do {
       do {
           status = fscanf (input_file, "%c", line_buf);
          } while (!leading_blanks_skipped && (*line_buf
           == ' ') &&
                   (status == 1));
      leading_blanks_skipped = TRUE;
      *eof = (status != 1) || (*line_buf == 26);
      eoln = (*line_buf == '\n') || (*line_buf == '\r')
       || (*eof);
```

```
      if (!(eoln || *eof) && (!is_legal_char
(*line_buf)))
         {
          char_error = TRUE;
          fprintf (stderr, " Error - Illegal char found
             in triangle %u: %02X (Hex)\n",
tri_count,(unsigned char) *line_buf);
         }
      line_buf++;
   } while (!*eof && !eoln);
   *--line_buf = '\0';
   *error = *error || char_error;
} /* get_line */

/*******************************************************/
   local procedure
   swap_vert
/*******************************************************/

void swap_vert(x,y,z,user_info)
  float *x;
  float *y;
  float *z;
  USER_INFO_STRUC *user_info;
  {
  float temp;

  if (user_info-> axis == 1) {
    temp = *z;
    *z = *x;
    *x = temp;
    }
  else if (user_info-> axis == 2) {
    temp = *z;
    *z = *y;
    *y = temp;
    }
```

```
  };
/*******************************************************/
   local procedure
   get_input_tri_asc
/*******************************************************/

static void get_input_tri_asc (in_file, tri_ptr,
user_info, error)

   FILE            *in_file;
   INPUT_TRI_STRUC *tri_ptr;
   USER_INFO_STRUC *user_info;
   BOOLEAN         *error;

{
   UINT16 items_read;

   tri_ptr-> next_input_tri = NULL_INPUT_TRI;

   fscanf (in_file, "%*s %f %f %f", &tri_ptr->x1,
     &tri_ptr->y1, &tri_ptr->z1);
   swap_vert(&tri_ptr->x1,&tri_ptr->y1,&tri_ptr->z1,
user_info);

   fscanf (in_file, "%*s %f %f %f", &tri_ptr->x2,
&tri_ptr->y2, &tri_ptr->z2);

swap_vert(&tri_ptr->x2,&tri_ptr->y2,&tri_ptr->z2,user_in
fo);

   items_read = fscanf (in_file, "%*s %f %f %f",
&tri_ptr->x3, &tri_ptr->y3, &tri_ptr->z3);

swap_vert(&tri_ptr->x3,&tri_ptr->y3,&tri_ptr->z3,user_in
fo);

   *error = (items_read != 3);
```

```
   if (*error)
     if (feof(in_file))
        fprintf (stderr, "\n Error - Premature end of
input file at triangle %u\n",tri_count);
     else fprintf (stderr, "\n Error -  I/O error in
reading input file at triangle %u\n",tri_count);

} /* get_input_tri_asc */
/*******************************************************/
   local procedure
   update_min_max_values
/*******************************************************/
void update_min_max_values (tri_ptr, min_x, min_y,
min_z,
                                              max_x,
max_y, max_z)
   INPUT_TRI_STRUC *tri_ptr;
   float           *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;

{
   if (tri_ptr-> x1 < *min_x)
      *min_x = tri_ptr-> x1;
   if (tri_ptr-> x2 < *min_x)
      *min_x = tri_ptr-> x2;
   if (tri_ptr-> x3 < *min_x)
      *min_x = tri_ptr-> x3;

   if (tri_ptr-> y1 < *min_y)
      *min_y = tri_ptr-> y1;
   if (tri_ptr-> y2 < *min_y)
      *min_y = tri_ptr-> y2;
   if (tri_ptr-> y3 < *min_y)
      *min_y = tri_ptr-> y3;

   if (tri_ptr-> z1 < *min_z)
      *min_z = tri_ptr-> z1;
```

```
    if (tri_ptr-> z2 < *min_z)
       *min_z = tri_ptr-> z2;
    if (tri_ptr-> z3 < *min_z)
       *min_z = tri_ptr-> z3;

    if (tri_ptr-> x1 > *max_x)
       *max_x = tri_ptr-> x1;
    if (tri_ptr-> x2 > *max_x)
       *max_x = tri_ptr-> x2;
    if (tri_ptr-> x3 > *max_x)
       *max_x = tri_ptr-> x3;

    if (tri_ptr-> y1 > *max_y)
       *max_y = tri_ptr-> y1;
    if (tri_ptr-> y2 > *max_y)
       *max_y = tri_ptr-> y2;
    if (tri_ptr-> y3 > *max_y)
       *max_y = tri_ptr-> y3;

    if (tri_ptr-> z1 > *max_z)
       *max_z = tri_ptr-> z1;
    if (tri_ptr-> z2 > *max_z)
       *max_z = tri_ptr-> z2;
    if (tri_ptr-> z3 > *max_z)
       *max_z = tri_ptr-> z3;

} /* update_min_max_values */

static void check_normal(i,j,k)
  float i,j,k;
  {
  float len;

  len = i*i+j*j+k*k;

  if ((len != 0) && ((len < .9) || (len > 1.1))) {
    fprintf(stderr,"\n Error reading normal for triangle
```

```
        %u.\n",tri_count);
      fprintf(stderr,"  Normal %g %g %g not a unit
        normal.\n",i,j,k);
      exit(SYS_FAILURE);
      };
    };

static void set_normal_info(i,j,k,normal_info)
  float i,j,k;
  UINT8 *normal_info;
  {
  if ((i > -NORMAL_MIN) && (i < NORMAL_MIN)) i = 0.0;
  if ((j > -NORMAL_MIN) && (j < NORMAL_MIN)) j = 0.0;
  if ((k > -NORMAL_MIN) && (k < NORMAL_MIN)) k = 0.0;

  *normal_info = 0;

  if (i > 0.0)        *normal_info |= NORMAL_X_POSITIVE;
  else if (i < 0.0)   *normal_info |= NORMAL_X_NEGATIVE;

  if (j > 0.0)        *normal_info |= NORMAL_Y_POSITIVE;
  else if (j < 0.0)   *normal_info |= NORMAL_Y_NEGATIVE;

  if (k > 0.0)        *normal_info |= NORMAL_Z_POSITIVE;
  else if (k < 0.0)   *normal_info |= NORMAL_Z_NEGATIVE;

  };

static void
get_normal_asc(line_buff,normal_info,user_info)
  STRING          *line_buff;
  UINT8           *normal_info;
  USER_INFO_STRUC *user_info;
  {
  INT16 items_read;
  INT16 items_expected = 3;
  float normal_i,normal_j,normal_k;
```

```
  items_read = sscanf(line_buff,"%*s %*s %f %f
%f",&normal_i,&normal_j,&normal_k);
  check_normal(normal_i,normal_j,normal_k);

  swap_vert(&normal_i,&normal_j,&normal_k,user_info);

  if (items_read != items_expected) {
    fprintf(stderr,"\n Error - I/O error trying to read
     normal in triangle %u\n",tri_count);
    exit(SYS_FAILURE);
    }
set_normal_info(normal_i,normal_j,normal_k,normal_info);

  };

/*******************************************************/
   exported procedure
   pfp_get_input_tri_list
/*******************************************************/

void pfp_get_input_tri_list (in_file, tri_list,
                             min_x, min_y, min_z,
                             max_x, max_y, max_z,
user_info)

   FILE             *in_file;
   INPUT_TRI_STRUC **tri_list;
   float            *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;
   USER_INFO_STRUC *user_info;

{
   STRING           line_buf[200];
   BOOLEAN          error = FALSE;
   BOOLEAN          eof = FALSE;
   UINT8            normal_info;
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
```

```
   *tri_list = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));
   tri_ptr = *tri_list;
   get_line (in_file, line_buf, &error, &eof);
   while (!eof && !error)
      {
       while (!eof && !error && (tolower(*line_buf) !=
NORMAL_FLAG_CHAR))
           get_line (stdin, line_buf, &error, &eof);

       if (!eof && !error)
          {
           tri_count++;

get_normal_asc(line_buf,&normal_info,user_info);
           get_line(stdin,line_buf,&error,&eof);
           get_input_tri_asc (stdin, tri_ptr, user_info,
&error);
           tri_ptr-> normal_info = normal_info;
           if (!error)
              {
               update_min_max_values (tri_ptr, min_x,
                 min_y, min_z,
                                                  max_x,
max_y, max_z);
                tri_ptr-> next_input_tri =
(INPUT_TRI_STRUC *) mem_alloc
                                                 (sizeof
(INPUT_TRI_STRUC));
                back_ptr = tri_ptr;
                tri_ptr = tri_ptr-> next_input_tri;
                get_line (stdin, line_buf, &error, &eof);
               }
          }
      }

   if (error)
```

```
        exit (SYS_FAILURE);

    if (tri_count > 0)
        {
         back_ptr-> next_input_tri = NULL_INPUT_TRI;
         mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
        }
    else
        {
         fprintf (stderr, "\n Error - No triangles found
in input file");
         exit (SYS_FAILURE);
        }

} /* pfp_get_input_tri_list */

/*******************************************************/
    local procedure
    get_normal_bin
/*******************************************************/

get_normal_bin(in_file,tri_ptr,user_info)
  FILE            *in_file;
  INPUT_TRI_STRUC *tri_ptr;
  USER_INFO_STRUC *user_info;
  {
  float normal_i,normal_j,normal_k;

  fread (&normal_i, sizeof(float), 1, in_file);
  fread (&normal_j, sizeof(float), 1, in_file);
  fread (&normal_k, sizeof(float), 1, in_file);
  check_normal(normal_i,normal_j,normal_k);
  swap_vert(&normal_i,&normal_j,&normal_k,user_info);

  set_normal_info(normal_i,normal_j,normal_k,&tri_ptr->
  normal_info);
```

```
return (normal_i != 0.0 || normal_j != 0.0 || normal_k
!= 0.0);

}/* get_normal_bin */

/*******************************************************/
   local procedure
   get_input_tri_bin
/*******************************************************/

void get_input_tri_bin (in_file, tri_ptr, user_info,
  error)

   FILE            *in_file;
   INPUT_TRI_STRUC *tri_ptr;
   USER_INFO_STRUC *user_info;
   BOOLEAN         *error;

{
   INT16 items_read;
   tri_ptr-> next_input_tri = NULL_INPUT_TRI;
   items_read = fread (tri_ptr, sizeof(float),
9,in_file);
   swap_vert(&tri_ptr->x1, &tri_ptr->y1, &tri_ptr->z1,
user_info);
   swap_vert(&tri_ptr->x2, &tri_ptr->y2, &tri_ptr->z2,
user_info);
   swap_vert(&tri_ptr->x3, &tri_ptr->y3, &tri_ptr->z3,
user_info);

   *error = (items_read != 9);

   if (*error) {
     if (feof(in_file))
       fprintf (stderr, "\n Error - Premature end of
input file at triangle %u\n",tri_count);
     else
```

```
        fprintf (stderr, "\n Error - I/O error reading
verticies in triangle %uinvalid format in binary input
file\n");
     exit (SYS_FAILURE);
     }

} /* get_input_tri_bin */

/*******************************************************/
   exported procedure
   pfp_get_input_tri_list_bin
/*******************************************************/

void pfp_get_input_tri_list_bin (in_file, tri_list,
                                 min_x, min_y, min_z,
                                 max_x, max_y, max_z,
                                 user_info)

   FILE            *in_file;
   INPUT_TRI_STRUC **tri_list;
   float           *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;
   USER_INFO_STRUC *user_info;

{
   BOOLEAN         error = FALSE;
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
   STRING          record[81];
   unsigned long   facet_records;
   UINT16          attrib_size;

   *tri_list = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));

   tri_ptr = *tri_list;
   record[80] = '\0';
```

```
   while (!error && (fread(record,80,1,in_file) > 0)) if
(user_info->attrib) {
     fread(&facet_records,4,1,in_file);
     while (!error && (facet_records != 0)) {
       tri_count++;
       facet_records--;
       get_normal_bin (in_file,tri_ptr,user_info);
       get_input_tri_bin (in_file, tri_ptr, user_info,
&error);
       if (!error) {
         fread(&attrib_size,2,1,in_file);
         while (attrib_size != 0) {
           attrib_size--;
           fread(record,1,1,in_file);
           }
         update_min_max_values (tri_ptr, min_x, min_y,
min_z, max_x, max_y, max_z);
         tri_ptr-> next_input_tri = (INPUT_TRI_STRUC *)
mem_alloc (sizeof (INPUT_TRI_STRUC));
         back_ptr = tri_ptr;
         tri_ptr = tri_ptr-> next_input_tri;
         }
       }
     }
   else while (!error && get_normal_bin
(in_file,tri_ptr,user_info)) {
     tri_count++;
     get_input_tri_bin (in_file, tri_ptr, user_info,
&error);
     if (!error) {
       fread(&attrib_size,2,1,in_file);
       while (attrib_size > 0) {
         fread(record,1,1,in_file);
         attrib_size--;
         }
       update_min_max_values (tri_ptr, min_x, min_y,
min_z, max_x, max_y, max_z);
```

```
        tri_ptr-> next_input_tri = (INPUT_TRI_STRUC *)
mem_alloc (sizeof (INPUT_TRI_STRUC));
        back_ptr = tri_ptr;
        tri_ptr = tri_ptr-> next_input_tri;
        }
      }
    if (error)
      exit (SYS_FAILURE);

    if (tri_count > 0) {
      back_ptr-> next_input_tri = NULL_INPUT_TRI;
      mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
      }
    else {
      fprintf (stderr, "\n Error - No triangles found in
input file");
      exit (SYS_FAILURE);
      }
    } /* pfp_get_input_tri_list_bin */
```

```
/*******************************************************/
/*               3D Systems Slice Program              */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPSEG.C                                            */
/*                                                      */
/* PFP Segment utility module                          */
/*******************************************************/

#include <stdio.h>
#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"

/*******************************************************/
   exported procedure
   Output_Error
/*******************************************************/

void Output_Error()
  {
  fprintf(stderr,"\n*** I/O Error in Slice output file
***\n");
  exit(SYS_FAILURE);
  };
/*******************************************************/
   exported procedure
   pfp_print_seg_list
/*******************************************************/

void pfp_print_seg_list (seg_list, z, seg_file,
section_type,
                         user_info)

   OUT_SEG_STRUC   **seg_list;
   UINT16          z;
```

```
   FILE             *seg_file;
   UINT16           section_type;
   USER_INFO_STRUC *user_info;

{
   OUT_SEG_STRUC *seg_ptr,*temp_ptr;

   if (section_type == LB_CODE) {
      if (fprintf(seg_file, "%s %u\n", L_TEXT, z)!=0)
Output_Error;
      if (*seg_list != NULL_SEG)
        if (fprintf(seg_file, "%s\n", LB_TEXT)!=0)
Output_Error;
      }

   if ((*seg_list != NULL_SEG) &&
       ((*seg_list)-> normal_info == section_type)) {
     if (!user_info-> section_codes_off)
        switch (section_type)
           {
            case (LH_CODE) :
               if (fprintf(seg_file, "%s\n",
LH_TEXT)!=0) Output_Error;
               break;
            case (NFDB_CODE) :
               if (fprintf(seg_file, "%s\n",
NFDB_TEXT)!=0) Output_Error;
               break;
            case (NFDH_CODE) :
               if (fprintf(seg_file, "%s\n",
NFDH_TEXT)!=0) Output_Error;
               break;
            case (NFDF_CODE) :
               if (fprintf(seg_file, "%s\n",
NFDF_TEXT)!=0) Output_Error;
               break;
            case (NFUB_CODE):
```

```
                if (fprintf(seg_file, "%s\n",
NFUB_TEXT)!=0) Output_Error;
                   break;
             case (NFUH_CODE):
                   if (fprintf(seg_file, "%s\n",
NFUH_TEXT)!=0) Output_Error;
                   break;
             case (NFUF_CODE) :
                   if (fprintf(seg_file, "%s\n",
NFUF_TEXT)!=0) Output_Error;
                   break;
             case (FUB_CODE):
                   if (fprintf(seg_file, "%s\n",
FUB_TEXT)!=0) Output_Error;
                   break;
             case (FUH_CODE):
                   if (fprintf(seg_file, "%s\n",
FUH_TEXT)!=0) Output_Error;
                   break;
             case (FUF_CODE):
                   if (fprintf(seg_file, "%s\n",
FUF_TEXT)!=0) Output_Error;
                   break;
             case (FDB_CODE) :
                   if (fprintf(seg_file, "%s\n",
FDB_TEXT)!=0) Output_Error;
                   break;
             case (FDH_CODE) :
                   if (fprintf(seg_file, "%s\n",
FDH_TEXT)!=0) Output_Error;
                   break;
             case (FDF_CODE) :
                   if (fprintf(seg_file, "%s\n",
FDF_TEXT)!=0) Output_Error;
                   break;
          }
```

```
      seg_ptr = *seg_list;

      while ((seg_ptr != NULL_SEG) &&
             (seg_ptr-> normal_info == section_type)) {
        if (fprintf(seg_file, " %u %u %u %u\n", seg_ptr->
x_1, seg_ptr-> y_1,
          seg_ptr-> x_2, seg_ptr-> y_2)!=0) Output_Error;
        temp_ptr = seg_ptr;
        seg_ptr = seg_ptr-> next_seg;
        mem_free(temp_ptr,sizeof(OUT_SEG_STRUC));
        }
      *seg_list = NULL_SEG;
      }

} /* pfp_print_seg_list */

/*******************************************************/
   exported procedure
   pfp_display_seg_list
/*******************************************************/

void pfp_display_seg_list (seg_list, display_mode)

   OUT_SEG_STRUC *seg_list;
   INT16         display_mode;
{
   OUT_SEG_STRUC *seg_ptr;
   set_display_mode (display_mode);
   seg_ptr = seg_list;
   while (seg_ptr != NULL_SEG)
      {
       draw_line (seg_ptr-> x_1, seg_ptr-> y_1,
seg_ptr-> x_2, seg_ptr-> y_2);
       seg_ptr = seg_ptr-> next_seg;
      }
} /* pfp_display_seg_list */
```

```
/*********************************************************/
   exported procedure
   pfp_delete_seg_list
/*********************************************************/

void pfp_delete_seg_list (seg_list)

   OUT_SEG_STRUC **seg_list;

{
   OUT_SEG_STRUC *next_ptr,*seg_ptr;
   seg_ptr = *seg_list;
   while (seg_ptr != NULL_SEG)
      {
       next_ptr = seg_ptr-> next_seg;
       mem_free (seg_ptr,sizeof(OUT_SEG_STRUC));
       seg_ptr = next_ptr;
      }
   *seg_list = seg_ptr;
} /* pfp_delete_seg_list */

/*********************************************************/
   swap_points
/*********************************************************/

static void swap_points (seg_ptr)

   OUT_SEG_STRUC *seg_ptr;

{
   UINT16       temp;

   temp = seg_ptr-> x_1;
   seg_ptr-> x_1 = seg_ptr-> x_2;
   seg_ptr-> x_2 = temp;
```

```
   temp = seg_ptr-> y_1;
   seg_ptr-> y_1 = seg_ptr-> y_2;
   seg_ptr-> y_2 = temp;

} /* swap_points */

/*********************************************************/
   local procedure
   get_next_seg
/*********************************************************/

static void get_next_seg (seg_ptr, seg_list,
next_seg_ptr)

   OUT_SEG_STRUC *seg_ptr;
   OUT_SEG_STRUC **seg_list;
   OUT_SEG_STRUC **next_seg_ptr;

{
   OUT_SEG_STRUC *list_ptr, *back_ptr;
   OUT_SEG_STRUC head_rec;
   BOOLEAN       seg_found = FALSE;

   *next_seg_ptr = NULL_SEG;
   back_ptr = &head_rec;
   head_rec .next_seg = *seg_list;
   list_ptr = *seg_list;

   while ((list_ptr != NULL_SEG) && !seg_found)
      {
       seg_found = (((seg_ptr-> x_1 == list_ptr->
         x_2) &&
       (seg_ptr-> y_1 == list_ptr-> y_2)) ||
       ((seg_ptr-> x_1 == list_ptr-> x_1) &&
       (seg_ptr-> y_1 == list_ptr-> y_1)));

       if (!seg_found)
```

```
                {
                 back_ptr = list_ptr;
                 list_ptr = list_ptr-> next_seg;
                }
            }

        if (seg_found)
            {
             if (!((seg_ptr-> x_1 == list_ptr-> x_2) &&
                   (seg_ptr-> y_1 == list_ptr-> y_2)))
                swap_points (list_ptr);

             back_ptr-> next_seg = list_ptr-> next_seg;
             *next_seg_ptr = list_ptr;
            }

        *seg_list = head_rec .next_seg;

    } /* get_next_seg */

    /*******************************************************/
        exported procedure
        pfp_sort_seg_list
    /*******************************************************/

    void pfp_sort_seg_list (list_1_ptr, loop_count)

        OUT_SEG_STRUC **list_1_ptr;
        INT16         *loop_count;

    {
        OUT_SEG_STRUC *list_2_ptr, *seg_ptr, *next_seg_ptr;

        seg_ptr = NULL_SEG;
        next_seg_ptr = NULL_SEG;
        list_2_ptr = NULL_SEG;
```

```
   *loop_count = 0;

   while (*list_1_ptr != NULL_SEG)
      {
       (*loop_count)++;
       seg_ptr = *list_1_ptr;
       *list_1_ptr = (*list_1_ptr)-> next_seg;
       seg_ptr-> next_seg = list_2_ptr;
       list_2_ptr = seg_ptr;
       do
          {
           get_next_seg (seg_ptr, list_1_ptr,
&next_seg_ptr);
           if (next_seg_ptr != NULL_SEG)
              {
               next_seg_ptr-> next_seg = list_2_ptr;
               list_2_ptr = next_seg_ptr;
               seg_ptr = next_seg_ptr;
              }
          } while (next_seg_ptr != NULL_SEG);
      }

   *list_1_ptr = list_2_ptr;

} /* pfp_sort_seg_list */

/*******************************************************/
   procedure
   append_seg_list
/*******************************************************/

void append_seg_list (list1, list2)

   OUT_SEG_STRUC **list1;
   OUT_SEG_STRUC **list2;

{
```

```
    OUT_SEG_STRUC *list1_ptr;

    if (*list1 == NULL_SEG)
       *list1 = *list2;
    else
       {
        list1_ptr = *list1;
        while (list1_ptr-> next_seg != NULL_SEG)
           list1_ptr = list1_ptr-> next_seg;
        list1_ptr-> next_seg = *list2;
       }
    *list2 = NULL_SEG;

} /* append_seg_list */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPSEG.H                                          */
/*                                                   */
/* PFP segment header file                           */
/*****************************************************/

#define   NORMAL_X_POSITIVE    1
#define   NORMAL_X_NEGATIVE    2
#define   NORMAL_Y_POSITIVE    4
#define   NORMAL_Y_NEGATIVE    8
#define   NORMAL_Z_POSITIVE   16
#define   NORMAL_Z_NEGATIVE   32
#define   SPARE1              64
#define   SPARE 2            128

struct out_seg
   {
    UINT16         x_1;
    UINT16         y_1;
    UINT16         x_2;
    UINT16         y_2;
    UINT8          normal_info;
    struct out_seg *next_seg;

   };
typedef struct out_seg OUT_SEG_STRUC;
#define NULL_SEG        (OUT_SEG_STRUC *) NULL
#define L_TEXT          "L"    /* Layer Header */
#define LB_TEXT         "LB"   /* border */
#define LH_TEXT         "LH"   /* border hatching */
#define NFDB_TEXT       "NFDB" /* Near-Flat Down skin
                                  Boarder */
#define NFDH_TEXT       "NFDH" /* Near-Flat Down skin
Hatch */
```

```
#define NFDF_TEXT       "NFDF" /* Near-Flat Down skin
                        Fill */
#define NFUB_TEXT       "NFUB" /* Near-Flat Up skin
                        Border */
#define NFUH_TEXT       "NFUH" /* Near-Flat Up skin
                        Border */
#define NFUF_TEXT       "NFUF" /* Near-Flat Up skin
                        Fill */
#define FDB_TEXT        "FDB"  /* Flat Down skin Fill */
#define FDH_TEXT        "FDH"  /* Flat Down skin Fill */
#define FDF_TEXT        "FDF"  /* Flat Down skin Fill */
#define FUB_TEXT        "FUB"  /* Flat Down skin Fill */
#define FUH_TEXT        "FUH"  /* Flat Down skin Fill */
#define FUF_TEXT        "FUF"  /* Flat Down skin Fill */
```

```
/*******************************************************/
/*              3D Systems Slice Program               */
/*         Copyright (C) 1988 3D Systems Inc.          */
/*                                                     */
/* PFPTRI.C                                            */
/*                                                     */
/* PFP Process Triangles Section                       */
/*******************************************************/

#include <stdio.h>
#include <math.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"

extern  INPUT_TRI_STRUC   *tri_bucket_array[];
extern  OUT_SEG_STRUC     *FD_seg_bucket_array[];
extern  OUT_SEG_STRUC     *FU_seg_bucket_array[];
extern  OUT_SEG_STRUC     *NFU_seg_bucket_array[];
extern  OUT_SEG_STRUC     *NFD_seg_bucket_array[];
extern  UINT16            z_control_table[];

/*******************************************************/
    local procedure
    add_near_flat_segment
    Adds a segment list
/*******************************************************/

static void add_near_flat_segment(segment,zi)
  OUT_SEG_STRUC **segment;
  UINT16        zi;
  {
  OUT_SEG_STRUC *seg_ptr;
```

```
  seg_ptr = *segment;
  *segment = NULL_SEG;

  if (seg_ptr->normal_info&NORMAL_Z_POSITIVE) {
    seg_ptr->next_seg = NFU_seg_bucket_array[zi];
    NFU_seg_bucket_array[zi] = seg_ptr;
    }
  else {
    seg_ptr->next_seg = NFD_seg_bucket_array[zi];
    NFD_seg_bucket_array[zi] = seg_ptr;
    };
  };

/*******************************************************/
    procedure
    process_flat_tri_list
    Takes each flat triangle and generates segments for
     z level and adds to FU or FD segment lists
/*******************************************************/

void process_flat_tri_list(flat_tri_list)
  INPUT_TRI_STRUC *flat_tri_list;
  {
  OUT_SEG_STRUC   *seg_list;
  INPUT_TRI_STRUC *temp_tri_ptr;
  UINT16          flat_tri_count = 0;
  UINT16          zi;

  while (flat_tri_list != NULL_INPUT_TRI) {
    flat_tri_count++;
    if (flat_tri_count%10 == 0)
      fprintf(stderr," processing flat triangles
(%u)\r",flat_tri_count);

    seg_list = NULL_SEG;
    pfp_convert_tri_to_segs(flat_tri_list,&seg_list);
    zi = get_z_binarily((UINT16) flat_tri_list->z1);
```

```
    if (flat_tri_list->normal_info&NORMAL_Z_POSITIVE) {

append_seg_list(&seg_list,&FU_seg_bucket_array[zi]);
      FU_seg_bucket_array[zi] = seg_list;
      }
    else {

append_seg_list(&seg_list,&FD_seg_bucket_array[zi]);
      FD_seg_bucket_array[zi] = seg_list;
      };

    temp_tri_ptr = flat_tri_list;
    flat_tri_list = flat_tri_list->next_input_tri;
    mem_free(temp_tri_ptr,sizeof(INPUT_TRI_STRUC));
    };
    fprintf(stderr," processing flat triangles
(%u)\n",flat_tri_count);
  };

/*******************************************************/
    procedure
    process_near_flat_tri_list
    Takes each near flat triangle and breaks into
      polygon for each z level and adds to NFU or NFD
      segment lists

/*******************************************************/

void process_near_flat_tri_list(near_flat_tri_list)
  INPUT_TRI_STRUC *near_flat_tri_list;
  {
  OUT_SEG_STRUC
*side1_seg,*side2_seg,*top_seg,*bot_seg;
  INPUT_TRI_STRUC *temp_tri_ptr;
  VECT_TRI_STRUC  vect_tri;
  UINT16          near_tri_count = 0;
  UINT16          zi;
```

```
  top_seg = NULL_SEG;
  bot_seg = NULL_SEG;

  while (near_flat_tri_list != NULL_INPUT_TRI) {
    near_tri_count++;
    if (near_tri_count%10 == 0)
      fprintf(stderr," processing near flat triangles
(%u)\r",near_tri_count);

pfp_convert_tri_to_vect(near_flat_tri_list,&vect_tri);
    zi = get_z_binarily((UINT16) vect_tri.vert1_z);

    while (z_control_table[zi+1] <= vect_tri.vert3_z) {

pfp_get_intersection_segment(&vect_tri,z_control_table[z
i],  &bot_seg);

pfp_get_intersection_segment(&vect_tri,z_control_table[z
i+1],&top_seg);
      side1_seg = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
      side1_seg->x_1 = bot_seg->x_1;
      side1_seg->y_1 = bot_seg->y_1;
      side1_seg->x_2 = top_seg->x_1;
      side1_seg->y_2 = top_seg->y_1;
      side2_seg = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
      side2_seg->x_1 = bot_seg->x_2;
      side2_seg->y_1 = bot_seg->y_2;
      side2_seg->x_2 = top_seg->x_2;
      side2_seg->y_2 = top_seg->y_2;
      side1_seg->normal_info = bot_seg->normal_info;
      side2_seg->normal_info = bot_seg->normal_info;
      /* Check for point segments in Top and Bottom. If
       so, at a vertex */
      if ((bot_seg->x_1 == bot_seg->x_2) &&
(bot_seg->y_1 == bot_seg->y_2)) {
```

```
        pfp_get_xy_normal_info((float)
side1_seg->x_1,(float) side1_seg->y_1,
                                (float)
side1_seg->x_2,(float) side1_seg->y_2,
                                (float) top_seg->x_2,
(float) top_seg->y_2,
                                &side1_seg->normal_info);
        pfp_get_xy_normal_info((float)
side2_seg->x_1,(float) side2_seg->y_1,
                                (float)
side2_seg->x_2,(float) side2_seg->y_2,
                                (float) top_seg->x_1,
(float) top_seg->y_1,
                                &side2_seg->normal_info);
        pfp_get_xy_normal_info((float)
top_seg->x_1,(float) top_seg->y_1,
                                (float)
top_seg->x_2,(float) top_seg->y_2,
                                (float)
bot_seg->x_1,(float) bot_seg->y_1,
                                &top_seg->normal_info);
        add_near_flat_segment(&top_seg,zi);
        }
      else {
        pfp_get_xy_normal_info((float)
side1_seg->x_1,(float) side1_seg->y_1,
                                (float)
side1_seg->x_2,(float) side1_seg->y_2,
                                (float) bot_seg->x_2,
(float) bot_seg->y_2,
                                &side1_seg->normal_info);
        pfp_get_xy_normal_info((float)
side2_seg->x_1,(float) side2_seg->y_1,
                                (float)
```

```
side2_seg->x_2,(float) side2_seg->y_2,
                                    (float) bot_seg->x_1,
(float) bot_seg->y_1,
                                    &side2_seg->normal_info);
        pfp_get_xy_normal_info((float)
bot_seg->x_1,(float) bot_seg->y_1,
                                    (float)
bot_seg->x_2,(float) bot_seg->y_2,
                                    (float)
top_seg->x_1,(float) top_seg->y_1,
                                    &bot_seg->normal_info);
        if ((top_seg->x_1 != top_seg->x_2) ||
(top_seg->y_1 != top_seg->y_2)) {
          pfp_get_xy_normal_info((float)
top_seg->x_1,(float) top_seg->y_1,
                                      (float)
top_seg->x_2,(float) top_seg->y_2,
                                      (float)
bot_seg->x_1,(float) bot_seg->y_1,
                                      &top_seg->normal_info);
          add_near_flat_segment(&top_seg,zi);
          };
        add_near_flat_segment(&bot_seg,zi);
        };
      add_near_flat_segment(&side1_seg,zi);
      add_near_flat_segment(&side2_seg,zi);
      zi++;
      };

    temp_tri_ptr = near_flat_tri_list;
    near_flat_tri_list =
near_flat_tri_list->next_input_tri;
    mem_free(temp_tri_ptr,sizeof(INPUT_TRI_STRUC));
    };

  if (top_seg != NULL_SEG)
    mem_free(top_seg,sizeof(OUT_SEG_STRUC));
```

```
  if (bot_seg != NULL_SEG)
    mem_free(bot_seg,sizeof(OUT_SEG_STRUC));

  fprintf(stderr," processing near flat triangles
(%u)\n",near_tri_count);
  };
```

```
/*******************************************************/
/*              3D Systems Slice Program               */
/*         Copyright (C) 1988 3D Systems Inc.          */
/*                                                     */
/* PFPTRI.H                                            */
/*                                                     */
/* PFP triangle definition header file                 */
/*******************************************************/

struct in_tri
   {
    float         x1;
    float         y1;
    float         z1;
    float         x2;
    float         y2;
    float         z2;
    float         x3;
    float         y3;
    float         z3;
    UINT8         normal_info;
    struct in_tri *next_input_tri;
   };

typedef struct in_tri INPUT_TRI_STRUC;

struct vector
   {
    float x;
    float y;
    float x_incr;
    float y_incr;
   };

struct vect_tri
   {
```

```
    struct vector   vector_1;
    struct vector   vector_2;
    struct vector   vector_3;
    float           vert1_z;
    float           vert2_z;
    float           vert3_z;
    UINT8           normal_info;
    struct vect_tri *next_tri;
   };

typedef struct vect_tri VECT_TRI_STRUC;

#define NULL_INPUT_TRI  (INPUT_TRI_STRUC *) NULL
#define NULL_VECT_TRI   (VECT_TRI_STRUC *) NULL
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPTUTL.C                                         */
/*                                                   */
/* PFP Triangle Utilities module                     */
/*****************************************************/

#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"

extern UINT16 z_control_table [];

/*****************************************************/
   exported procedure
   pfp_get_min_z
/*****************************************************/

void pfp_get_min_z (tri, min_z)

   INPUT_TRI_STRUC *tri;
   float           *min_z;

{
   *min_z = (float) (MAX_Z_VAL+1);

   if (tri-> z1 < *min_z)
      *min_z = tri-> z1;

   if (tri-> z2 < *min_z)
      *min_z = tri-> z2;
```

```
   if (tri-> z3 < *min_z)
      *min_z = tri-> z3;

} /* pfp_get_min_z */

/*******************************************************/
   local procedure
   get_max_z
/*******************************************************/

void get_max_z (tri, max_z)

   INPUT_TRI_STRUC *tri;
   float           *max_z;

{
   *max_z = 0.0;
   if (tri-> z1 > *max_z)
      *max_z = tri-> z1;

   if (tri-> z2 > *max_z)
      *max_z = tri-> z2;

   if (tri-> z3 > *max_z)
      *max_z = tri-> z3;

} /* get_max_z */

/*******************************************************/
   local procedure
   order_points
/*******************************************************/

void order_points (input_tri)

   INPUT_TRI_STRUC *input_tri;
```

```
{
   INPUT_TRI_STRUC *temp_tri;
   float           min_z;
   float           max_z;

   temp_tri = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));
   pfp_get_min_z (input_tri, &min_z);
   get_max_z (input_tri, &max_z);
   if ((input_tri-> z1 == max_z) && (input_tri-> z2 ==
min_z))
      {
       temp_tri-> x1 = input_tri-> x2;
       temp_tri-> y1 = input_tri-> y2;
       temp_tri-> z1 = input_tri-> z2;

       temp_tri-> x2 = input_tri-> x3;
       temp_tri-> y2 = input_tri-> y3;
       temp_tri-> z2 = input_tri-> z3;

       temp_tri-> x3 = input_tri-> x1;
       temp_tri-> y3 = input_tri-> y1;
       temp_tri-> z3 = input_tri-> z1;
      }

   else if ((input_tri-> z1 == max_z) && (input_tri-> z3
== min_z))
      {
       temp_tri-> x1 = input_tri-> x3;
       temp_tri-> y1 = input_tri-> y3;
       temp_tri-> z1 = input_tri-> z3;

       temp_tri-> x2 = input_tri-> x2;
       temp_tri-> y2 = input_tri-> y2;
       temp_tri-> z2 = input_tri-> z2;
```

```
        temp_tri-> x3 = input_tri-> x1;
        temp_tri-> y3 = input_tri-> y1;
        temp_tri-> z3 = input_tri-> z1;
       }

   else if ((input_tri-> z2 == max_z) && (input_tri-> z3
== min_z))
       {
        temp_tri-> x1 = input_tri-> x3;
        temp_tri-> y1 = input_tri-> y3;
        temp_tri-> z1 = input_tri-> z3;

        temp_tri-> x2 = input_tri-> x1;
        temp_tri-> y2 = input_tri-> y1;
        temp_tri-> z2 = input_tri-> z1;

        temp_tri-> x3 = input_tri-> x2;
        temp_tri-> y3 = input_tri-> y2;
        temp_tri-> z3 = input_tri-> z2;
       }

   else if ((input_tri-> z2 == max_z) && (input_tri-> z1
== min_z))
       {
        temp_tri-> x1 = input_tri-> x1;
        temp_tri-> y1 = input_tri-> y1;
        temp_tri-> z1 = input_tri-> z1;

        temp_tri-> x2 = input_tri-> x3;
        temp_tri-> y2 = input_tri-> y3;
        temp_tri-> z2 = input_tri-> z3;

        temp_tri-> x3 = input_tri-> x2;
        temp_tri-> y3 = input_tri-> y2;
        temp_tri-> z3 = input_tri-> z2;
       }
```

```
   else if ((input_tri-> z3 == max_z) && (input_tri-> z1
== min_z))
      {
       temp_tri-> x1 = input_tri-> x1;
       temp_tri-> y1 = input_tri-> y1;
       temp_tri-> z1 = input_tri-> z1;

       temp_tri-> x2 = input_tri-> x2;
       temp_tri-> y2 = input_tri-> y2;
       temp_tri-> z2 = input_tri-> z2;

       temp_tri-> x3 = input_tri-> x3;
       temp_tri-> y3 = input_tri-> y3;
       temp_tri-> z3 = input_tri-> z3;
      }

   else
      {
       temp_tri-> x1 = input_tri-> x2;
       temp_tri-> y1 = input_tri-> y2;
       temp_tri-> z1 = input_tri-> z2;

       temp_tri-> x2 = input_tri-> x1;
       temp_tri-> y2 = input_tri-> y1;
       temp_tri-> z2 = input_tri-> z1;

       temp_tri-> x3 = input_tri-> x3;
       temp_tri-> y3 = input_tri-> y3;
       temp_tri-> z3 = input_tri-> z3;
      }

   temp_tri-> normal_info = input_tri-> normal_info;
   temp_tri-> next_input_tri = input_tri->
next_input_tri;
   copy(temp_tri, input_tri, sizeof (INPUT_TRI_STRUC));
   mem_free(temp_tri,sizeof(INPUT_TRI_STRUC));
```

```
} /* order_points */

/*********************************************************/
   local procedure
   get_vector
/*********************************************************/

void get_vector (vector_ptr, start_x, start_y, start_z,
                        end_x, end_y, end_z)

   struct vector *vector_ptr;
   float start_x;
   float start_y;
   float start_z;
   float end_x;
   float end_y;
   float end_z;

{
   double delta_x, delta_y, delta_z;

   vector_ptr-> x = start_x;
   vector_ptr-> y = start_y;

   delta_z = end_z - start_z;

   if (delta_z > 0.0)
      {
       delta_x = end_x - start_x;
       delta_y = end_y - start_y;
       vector_ptr-> x_incr = delta_x / delta_z;
       vector_ptr-> y_incr = delta_y / delta_z;
      }
   else
      {
       vector_ptr-> x_incr = 0;
       vector_ptr-> y_incr = 0;
```

```
        }

    } /* get_vector */

    /*******************************************************/
       exported procedure
       pfp_get_intersection_segment
    /*******************************************************/
    void pfp_get_intersection_segment (vect_ptr, z_value,
    seg_ptr)

       VECT_TRI_STRUC *vect_ptr;
       UINT16         z_value;
       OUT_SEG_STRUC  **seg_ptr;

    {
       OUT_SEG_STRUC  *segment;

       if (*seg_ptr == NULL_SEG)
         *seg_ptr = (OUT_SEG_STRUC *)
    mem_alloc(sizeof(OUT_SEG_STRUC));
       segment = *seg_ptr;

       segment-> normal_info = vect_ptr-> normal_info;
       segment-> x_1 = round_flt(vect_ptr-> vector_1 .x
                       + vect_ptr-> vector_1.x_incr *
    (z_value - vect_ptr->vert1_z));
       segment-> y_1 = round_flt(vect_ptr-> vector_1 .y
                       + vect_ptr-> vector_1.y_incr *
    (z_value - vect_ptr->vert1_z));

       if (vect_ptr-> vert2_z > z_value)
          {
           segment-> x_2 = round_flt(vect_ptr-> vector_2 .x
                       + vect_ptr-> vector_2.x_incr *
    (z_value - vect_ptr->vert1_z));
           segment-> y_2 = round_flt(vect_ptr-> vector_2 .y
```

```
                    + vect_ptr-> vector_2.y_incr *
(z_value - vect_ptr->vert1_z));
      }
   else
      {
       segment-> x_2 = round_flt(vect_ptr-> vector_3 .x
                    + vect_ptr-> vector_3.x_incr *
(z_value - vect_ptr->vert2_z));
       segment-> y_2 = round_flt(vect_ptr-> vector_3 .y
                    + vect_ptr-> vector_3.y_incr *
(z_value - vect_ptr->vert2_z));
      }

} /* pfp_get_intersection_segment */

/*******************************************************/
   exported procedure
/*******************************************************/

void pfp_convert_tri_to_vect(input_tri, vect_tri)

   INPUT_TRI_STRUC *input_tri;
   VECT_TRI_STRUC  *vect_tri;
{
   struct vector   *vector_ptr;
   order_points (input_tri);
   vector_ptr = (struct vector *) vect_tri;
   get_vector (vector_ptr++,
               input_tri-> x1, input_tri-> y1,
input_tri-> z1,
               input_tri-> x3, input_tri-> y3,
input_tri-> z3);

   get_vector (vector_ptr++,
               input_tri-> x1, input_tri-> y1,
input_tri-> z1,
```

```
                input_tri-> x2, input_tri-> y2,
input_tri-> z2);

   get_vector (vector_ptr,
                input_tri-> x2, input_tri-> y2,
input_tri-> z2,
                input_tri-> x3, input_tri-> y3,
input_tri-> z3);

   vect_tri-> vert1_z  = input_tri->z1;
   vect_tri-> vert2_z  = input_tri->z2;
   vect_tri-> vert3_z  = input_tri->z3;
   vect_tri-> normal_info = input_tri-> normal_info;
   vect_tri-> next_tri = NULL_VECT_TRI;

} /* pfp_convert_tri_to_vect */

/*******************************************************/
   procedure
/*******************************************************/

void pfp_convert_tri_to_segs (tri_ptr, seg_list)

   INPUT_TRI_STRUC *tri_ptr;
   OUT_SEG_STRUC   **seg_list;

{
   OUT_SEG_STRUC   *seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
   seg_ptr-> x_1 = tri_ptr-> x1;
   seg_ptr-> y_1 = tri_ptr-> y1;
   seg_ptr-> x_2 = tri_ptr-> x2;
   seg_ptr-> y_2 = tri_ptr-> y2;
   seg_ptr->normal_info = tri_ptr->normal_info;
```

```
   pfp_get_xy_normal_info (tri_ptr-> x1, tri_ptr-> y1,
                           tri_ptr-> x2, tri_ptr-> y2,
                           tri_ptr-> x3, tri_ptr->
y3,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
   seg_ptr-> x_1 = tri_ptr-> x2;
   seg_ptr-> y_1 = tri_ptr-> y2;
   seg_ptr-> x_2 = tri_ptr-> x3;
   seg_ptr-> y_2 = tri_ptr-> y3;
   seg_ptr->normal_info = tri_ptr->normal_info;
   pfp_get_xy_normal_info (tri_ptr-> x2, tri_ptr-> y2,
                           tri_ptr-> x3, tri_ptr-> y3,
                           tri_ptr-> x1, tri_ptr->
y1,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
   seg_ptr-> x_1 = tri_ptr-> x3;
   seg_ptr-> y_1 = tri_ptr-> y3;
   seg_ptr-> x_2 = tri_ptr-> x1;
   seg_ptr-> y_2 = tri_ptr-> y1;
   seg_ptr->normal_info = tri_ptr->normal_info;
   pfp_get_xy_normal_info (tri_ptr-> x3, tri_ptr-> y3,
                           tri_ptr-> x1, tri_ptr-> y1,
                           tri_ptr-> x2, tri_ptr->
y2,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

} /* pfp_convert_tri_to_segs */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPUTL.C                                          */
/*                                                   */
/* PFP Utilities module                              */
/*****************************************************/

#include <stdio.h>
#include <time.h>
#include <malloc.h>
#include <math.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"

static long   last_time;

extern long max_mem_alloc;
extern long total_mem_alloc;
extern long curr_mem_alloc;

float round_flt(float_val)
  float float_val;
  {
  long int_val;

  int_val = float_val;
  if (((float) int_val+0.5) <= float_val) int_val++;
  return (int_val);

} /* round_float */

double round_dbl(dbl_val)
  double dbl_val;
```

```
  {
  if (dbl_val < 0.0) dbl_val = ceil(0.5+dbl_val);
  else dbl_val = floor(0.5+dbl_val);
  return (dbl_val);

} /* round_dbl */

INT16 sign(d)
  double d;
  {
  if (d < 0.0) return(-1);
  else return(1);
  };

STRING *mem_alloc (size)

   UINT16 size;

{
   STRING *ptr;

   ptr = (STRING *) malloc (size);

   curr_mem_alloc += size;
   total_mem_alloc += size;

   if (curr_mem_alloc > max_mem_alloc)
      max_mem_alloc = curr_mem_alloc;

   if (ptr == (STRING *) NULL)
      {
       fprintf (stderr, "\n error - memory allocation
failed\n");
       fprintf (stderr, "\n current allocation = %ld\n",
curr_mem_alloc);
       exit (SYS_FAILURE);
      }
```

```
   return (ptr);

} /* mem_alloc */

void mem_free (ptr,num_bytes)

   STRING  *ptr;
   UINT16  num_bytes;

{
   if (ptr != (STRING *) NULL)
      {
       curr_mem_alloc -= num_bytes;
       free (ptr);
      }
   else
      {
       fprintf (stderr, "\n error - null pointer freed
\n");
       exit (SYS_FAILURE);
      }
} /* mem_free */

void get_timestamp (timestamp)

   STRING *timestamp;

{
   long time_buf;

   time (&time_buf);
   strcpy (timestamp, ctime (&time_buf));

} /* get_timestamp */

void openfile (filename, file_ptr, open_mode)
```

```
   STRING *filename;
   FILE   **file_ptr;
   STRING *open_mode;

{
   if (*filename == '\0')
      *file_ptr = (FILE *) NULL;
   else
      {
       *file_ptr = fopen (filename, open_mode);
       if (*file_ptr == (FILE *) NULL)
          {
           fprintf (stderr, "\n error - unable to open
%s\n", filename);
           exit (SYS_FAILURE);
          }
      }

} /* openfile */

void copy (source_ptr, dest_ptr, byte_count)

   STRING *source_ptr;
   STRING *dest_ptr;
   INT16  byte_count;

{
   INT16 i;

   for (i = 0; i < byte_count; i++)
      {
       *dest_ptr = *source_ptr;
       dest_ptr++;
       source_ptr++;
      }

} /* copy */
```

```
void pfp_get_xy_normal_info
(x1,y1,x2,y2,x3,y3,normal_info)
  float x1;
  float y1;
  float x2;
  float y2;
  float x3;
  float y3;
  UINT8 *normal_info;
  {
  double slope, offset;

  *normal_info &=
(255-NORMAL_X_NEGATIVE-NORMAL_X_POSITIVE-
    NORMAL_Y_NEGATIVE-NORMAL_Y_POSITIVE);

  if ((x2 - x1) != 0.0) slope = (y2 - y1) / (x2 - x1);
  else slope = 0.0;

  offset = y1 - slope*x1;

  if (slope == 0.0) {
    if (y1 == y2) {
      if (y3 < y2) *normal_info |= NORMAL_Y_POSITIVE;
      else *normal_info |= NORMAL_Y_NEGATIVE;
      };
    if (x1 == x2) {
      if (x3 < x2) *normal_info |= NORMAL_X_POSITIVE;
      else *normal_info |= NORMAL_X_NEGATIVE;
      };
    }
  else {
    if (((y3 - offset)/slope) >= x3) *normal_info |=
NORMAL_X_POSITIVE;
    else *normal_info |= NORMAL_X_NEGATIVE;
    if ((slope*x3 + offset) >= y3) *normal_info |=
NORMAL_Y_POSITIVE;
```

```
    else *normal_info |= NORMAL_Y_NEGATIVE;
    }
};
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPUTL.H                                            */
/*                                                     */
/* PFP Utility Prototyes                               */
/*******************************************************/

double round_dbl();
float  round_flt();
STRING *mem_alloc();
INT16  sign();
```

```
/*********************************************************
/*                    3D Systems Slice Program           */
/*                Copyright (C) 1988 3D Systems Inc.     */
/*                                                       */
/* PFPZACT.C                                             */
/*                                                       */
/* PFP Z Active list module                              */
/*                                                       */
*********************************************************/

#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"


/*********************************************************
   exported procedure

   pfp_update_z_active_list


/*********************************************************

void  pfp_update_z_active_list  (z_bucket_ptr,  seg_list,
z_active_list)

   INPUT_TRI_STRUC **z_bucket_ptr;
   OUT_SEG_STRUC   **seg_list;
   VECT_TRI_STRUC  **z_active_list;

{
   VECT_TRI_STRUC  *new_vect_tri;
   INPUT_TRI_STRUC *z_bucket_back_ptr, **list_head;

   list_head = z_bucket_ptr;
```

```
   while (*z_bucket_ptr != NULL_INPUT_TRI) {
     new_vect_tri = (VECT_TRI_STRUC *) mem_alloc (sizeof
(VECT_TRI_STRUC));
     pfp_convert_tri_to_vect (*z_bucket_ptr, new_vect_tr-
i);
     new_vect_tri-> next_tri = *z_active_list;
     *z_active_list = new_vect_tri;
     z_bucket_back_ptr = *z_bucket_ptr;
     *z_bucket_ptr = (*z_bucket_ptr)-> next_input_tri;
     mem_free  (z_bucket_back_ptr,sizeof(INPUT_TRI_STRUC));
     };

   *list_head = NULL_INPUT_TRI;

} /* pfp_update_z_active_list */

/*********************************************************
   exported procedure

   pfp_process_z_active_list

/*********************************************************

void  pfp_process_z_active_list  (z_active_list,  z_value,
seg_list)

   VECT_TRI_STRUC **z_active_list;
   UINT16          z_value;
   OUT_SEG_STRUC   **seg_list;

{
   VECT_TRI_STRUC *tri_back_ptr, *tri_list_ptr;
   OUT_SEG_STRUC  *seg_ptr;
   VECT_TRI_STRUC list_head;

   z_value++;
   list_head .next_tri = *z_active_list;
```

```
   tri_list_ptr = *z_active_list;
   tri_back_ptr = &list_head;

   while (tri_list_ptr != NULL_VECT_TRI)
      {
       if (tri_list_ptr-> vert3_z <= z_value)
          {
           tri_back_ptr->   next_tri   =   tri_list_ptr->
next_tri;
           mem_free  (tri_list_ptr,sizeof(VECT_TRI_STRUC));
           tri_list_ptr = tri_back_ptr-> next_tri;
          }
       else
          {
           seg_ptr = (OUT_SEG_STRUC *) mem_alloc(sizeof
            (OUT_SEG_STRUC));
           pfp_get_intersection_segment    (tri_list_ptr,
z_value, &seg_ptr);
           seg_ptr-> next_seg = *seg_list;
           *seg_list = seg_ptr;
           tri_list_ptr = tri_list_ptr-> next_tri;
           tri_back_ptr = tri_back_ptr-> next_tri;
}
}

   *z_active_list = list_head .next_tri;

} /* pfp_process_z_active_list */
```

```
/*********************************************************
/*                    3D Systems Slice Program           */
/*                Copyright (C) 1988 3D Systems Inc.     */
/*                                                       */
/*  PRTUTL.C  -  Debugging Print Routines                */
/*                                                       */
/*                           MODIFICATION LOG            */
/*-------------------------------------------------------*/
/* date       who              what                      */
/*-------------------------------------------------------*/
/*          |                |                           */
/*          |                |                           */
/*********************************************************

#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfpchl.h"
#include "pfpcure.h"


void print_cure_hash_point(out_file,hash_point)
  FILE            *out_file;
  CURE_HASH_STRUC *hash_point;
  {
  fprintf(out_file,"Cure Hash Point %lu\n",hash_point);
  if (hash_point!=NULL_CURE_HASH) {
    fprintf(out_file,"X = %u\n",hash_point->x);
    fprintf(out_file,"Y = %u\n",hash_point->y);
    fprintf(out_file,"Segment Pointer = %lu\n",hash_point
     ->seg_ptr);
    fprintf(out_file,"Last Pointer    = %lu\n",hash_point
     ->last_hash);
    fprintf(out_file,"Next Pointer    = %lu\n\n",hash_
     point->next_hash);
    }
```

```
  fflush(out_file);
  };


void print_cure_hash(out_file,hash_list,hash_index)
  FILE            *out_file;
  CURE_HASH_STRUC *hash_list;
  UINT16          hash_index;
  {
  fprintf(out_file,"\n\n\nCure Hash Index %u:\n",hash_
     index);
  while (hash_list!=NULL_CURE_HASH) {
    print_cure_hash_point(out_file,hash_list);
    hash_list=hash_list->next_hash;
    };
  };


void print_hatch_point(out_file,point)
  FILE              *out_file;
  HATCH_POINT_STRUC *point;
  {
  if (point == NULL_HATCH_POINT) fprintf(out_file,"\nHatch
Point NULL\n");
  else {
    fprintf(out_file,"\nHatch Point %u %u\n",point->x_
     val,point->y_val);
    fprintf(out_file,"  angle %d enter_flag %d triangle
 number %u\n",point->angle,point->enter_flag,point->
     triangle_number);
    fprintf(out_file,"  next_point %lu\n",point->next_
     point);
    };
  fflush(out_file);
  };

void print_hatch_list(out_file,point_ptr)
```

```
FILE                *out_file;
HATCH_POINT_STRUC *point_ptr;
{
fprintf(out_file,"---------- BEGIN HATCH POINT LIST
   ----------\n");
while (point_ptr != (HATCH_POINT_STRUC *) NULL) {
  print_hatch_point(out_file,point_ptr);
  point_ptr = point_ptr->next_point;
  };
fprintf(out_file,"----------- END HATCH POINT LIST
   -----------\n");
};

oid print_normal_info(out_file,norm)
FILE                *out_file;
UINT8               norm;
{
fprintf (out_file, "\n");
fprintf (out_file, "NORMAL_X_POSITIVE = %s     ",
         norm & NORMAL_X_POSITIVE ? "True " : "False");
fprintf (out_file, "NORMAL_X_NEGATIVE = %s\n",
         norm & NORMAL_X_NEGATIVE ? "True " : "False");
fprintf (out_file, "NORMAL_Y_POSITIVE = %s     ",
         norm & NORMAL_Y_POSITIVE ? "True " : "False");
fprintf (out_file, "NORMAL_Y_NEGATIVE = %s\n",
         norm & NORMAL_Y_NEGATIVE ? "True " : "False");
fprintf (out_file, "NORMAL_Z_POSITIVE = %s     ",
         norm & NORMAL_Z_POSITIVE ? "True " : "False");
fprintf (out_file, "NORMAL_Z_NEGATIVE = %s\n",
         norm & NORMAL_Z_NEGATIVE ? "True " : "False");
fprintf (out_file, "SPARE1             = %s     ",
         norm & SPARE1 ? "True " : "False");
fprintf (out_file, "SPARE2             = %s     ",
         norm & SPARE2 ? "True " : "False");
fprintf (out_file, "\n");
};
```

```
void print_segment (out_file, seg)
   FILE          *out_file;
   OUT_SEG_STRUC *seg;
{
   fprintf (out_file, "\n Segment\n");
   fprintf (out_file, " x1 = %u  y1 = %u \n", seg-> x_1,
seg-> y_1);
   fprintf (out_file, " x2 = %u  y2 = %u \n", seg-> x_2,
seg-> y_2);
   print_normal_info(out_file,seg->normal_info);

   fprintf (out_file, " next ptr = %lu \n", seg-> next_
     seg);
   fflush(out_file);

} /* print_segment */

void print_seg_list (out_file, seg_list)

   FILE          *out_file;
   OUT_SEG_STRUC *seg_list;

{
   OUT_SEG_STRUC *seg_ptr;
   UINT16        list_length = 0;

   seg_ptr = seg_list;
   while (seg_ptr != NULL_SEG)
      {
       list_length++;
       seg_ptr = seg_ptr-> next_seg;
      }
```

```
   fprintf (out_file, "\nprinting segment list \n");
   fprintf (out_file, "list length = %u ", list_length);
   fprintf (out_file, "list =  \n");

   seg_ptr = seg_list;

   while (seg_ptr != NULL_SEG)
      {
       print_segment (out_file, seg_ptr);
       seg_ptr = seg_ptr-> next_seg;
      }
} /* print_seg_list */


void print_seg_bucket_array (out_file, seg_bucket_array)

   FILE          *out_file;
   OUT_SEG_STRUC **seg_bucket_array;

{
   OUT_SEG_STRUC *seg_ptr;
   UINT16        i;


   fprintf (out_file, "\n Z Segment Bucket Structure:\n");

   for (i = 0; i < MAX_Z_INDEX; i++) {
       seg_ptr = seg_bucket_array [i];
       if (seg_ptr != NULL_SEG)
          fprintf (out_file, "\n\n Z Bucket Index = %d
\n", i);
       while (seg_ptr != NULL_SEG) {
           print_segment (out_file, seg_ptr);
           seg_ptr = seg_ptr-> next_seg;
          }
       }
```

```
} /* print_seg_bucket_array */

void print_input_tri (out_file, tri)

   FILE            *out_file;
   INPUT_TRI_STRUC *tri;

{
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri-> x1,
tri-> y1, tri-> z1);
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri-> x2,
tri-> y2, tri-> z2);
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri-> x3,
tri-> y3, tri-> z3);
   print_normal_info(out_file,tri->normal_info);

} /* print_input_tri */


in_box (tri_ptr)

  INPUT_TRI_STRUC *tri_ptr;

{
   if ((tri_ptr-> z1 <= 3000) || (tri_ptr-> z2 <= 3000) ||

                                 (tri_ptr-> z3 <= 3000))
    {
     if ((tri_ptr-> z1 >= 3000) || (tri_ptr-> z2 >= 3000)
||
                                   (tri_ptr-> z3 >= 3000))

      {
       if (
           ((tri_ptr-> x1 < 20000) && (tri_ptr-> x1 >
13000)) ||
```

```
            ((tri_ptr-> x2 < 20000) && (tri_ptr-> x2 >
13000)) ||
            ((tri_ptr-> x3 < 20000) && (tri_ptr-> x3 >
13000))
           )
          {
           if (
               ((tri_ptr-> y1 < 55000) && (tri_ptr-> y1 >
48000)) ||
               ((tri_ptr-> y2 < 55000) && (tri_ptr-> y2 >
48000)) ||
               ((tri_ptr-> y3 < 55000) && (tri_ptr-> y3 >
48000))
              )
             return (1);
          }
        }
      }
    return (0);

} /* in_box */



void print_tri_list (out_file,tri_list)

  FILE            *out_file;
  INPUT_TRI_STRUC *tri_list;

{
   INPUT_TRI_STRUC *tri_ptr;
   UINT16           list_length = 0;


   tri_ptr = tri_list;
   while (tri_ptr != NULL_INPUT_TRI)
      {
```

```
        list_length++;
        tri_ptr = tri_ptr-> next_input_tri;
       }

    fprintf (out_file, "\n list length = %u ", list_
     length);
    fprintf (out_file, "\n list =  \n");

   tri_ptr = tri_list;
   while (tri_ptr != NULL_INPUT_TRI) {
/*if (in_box (tri_ptr)) */
     print_input_tri (out_file, tri_ptr);
     tri_ptr = tri_ptr-> next_input_tri;
     }

} /* print_tri_list */



void print_tri_bucket_array (out_file, z_bucket_array)

   FILE            *out_file;
   INPUT_TRI_STRUC **z_bucket_array;

{
   INPUT_TRI_STRUC *tri_ptr;
   INT16           i;


   fprintf (out_file, "\n Tri Bucket Structure:\n");

   for (i = 0; i < MAX_Z_INDEX; i++)
      {
       tri_ptr = z_bucket_array [i];
       if (tri_ptr != NULL_INPUT_TRI)
```

```
            fprintf (out_file, "\n\n Z Bucket Index = %d
\n", i);
          while (tri_ptr != NULL_INPUT_TRI)
             {
              print_input_tri (out_file, tri_ptr);
              tri_ptr = tri_ptr-> next_input_tri;
             }
         }

} /* print_tri_bucket_array */




void print_vect_tri (out_file, vect_tri)

   FILE           *out_file;
   VECT_TRI_STRUC *vect_tri;

{
   fprintf (out_file, " vector 1:\n");
   fprintf (out_file, "    x = %f   y = %f   x_incr = %f
 y_incr = %f\n",
                      vect_tri-> vector_1 .x,
                      vect_tri-> vector_1 .y,
                      vect_tri-> vector_1 .x_incr,
                      vect_tri-> vector_1 .y_incr);

   fprintf (out_file, " vector 2:\n");
   fprintf (out_file, "    x = %f   y = %f   x_incr = %f
 y_incr = %f\n",
                      vect_tri-> vector_2 .x,
                      vect_tri-> vector_2 .y,
                      vect_tri-> vector_2 .x_incr,
                      vect_tri-> vector_2 .y_incr);
```

```
   fprintf (out_file, " vector 3:\n");
   fprintf (out_file, "    x = %f   y = %f   x_incr = %f
y_incr = %f\n",
                      vect_tri-> vector_3 .x,
                      vect_tri-> vector_3 .y,
                      vect_tri-> vector_3 .x_incr,
                      vect_tri-> vector_3 .y_incr);

    fprintf    (out_file,    "Vertex    1    Z    =
%f\n",vect_tri->vert1_z);
    fprintf    (out_file,    "Vertex    2    Z    =
%f\n",vect_tri->vert2_z);
    fprintf    (out_file,    "Vertex    3    Z    =
%f\n",vect_tri->vert3_z);
   print_normal_info(out_file,vect_tri->normal_info);

} /* print_vect_tri */


void print_z_active_list (out_file, z_active_list)

   FILE           *out_file;
   VECT_TRI_STRUC *z_active_list;

{
   while (z_active_list != NULL_VECT_TRI)
      {
       print_vect_tri (out_file, z_active_list);
       z_active_list = z_active_list-> next_tri;
      }
   fflush (out_file);

} /* print_z_active_list */
```

```
{============================================================

   3D Systems Stereolithography System Software

============================================================

   POWER.PAS
   SLA-1 Power Control Program

------------------------------------------------------------

   Recent History:
   11/12/87  Ver 2.30  first Beta software release
    1/08/88  Ver 2.50  retitled program
    1/25/88  Ver 2.60  better messages
    2/08/88  -         converted from Turbo Basic to Turbo
Pascal Version 4.0
                       better Laser Board I/O
                       power status line display
                       support of toggling-Laser power
    2/26/88  Ver 2.62  fix for toggling Laser power
    3/09/88  Ver 2.80  added copyright notice
    4/04/88  Ver 3.00  change menus


------------------------------------------------------------
----}

uses Crt,Dos;

const
  VersionId = '3.00';
  LaserBoardAddress = $300;
  PowerPort     = $304;
  _INIT_OUT     = $01;  { outputs }
  _ZSTAGE_OUT   = $02;
  _SHUTTER_OUT  = $04;
```

```
  _LASER_OUT    = $08;
  _INIT_IN      = $10;  { inputs }
  _DOOR_IN      = $20;
  _SHUTTER_IN   = $40;
  _LASER_IN     = $80;
  Indent = 22;
  LASERVECTOR  = $64;
  STEREOVECTOR = $65;
  HomeDirectory = 'C:\3DSYS\';
  FixFileName   = 'FIX.DAT';
  ParamFileName = 'POWER.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  Ctrl-C = ^C;

type
  str4 = string[4];
  str16 = string[16];
  LaserPowerType = (_Unknown,_Level,_Toggle);
  OnOffType = (On,Off);

var
  key: Char;
  regs: Registers;
  LaserVersion,StereoVersion: str4;
  ParamFile,ErrorFile,FixFile: Text;
  temp,IndentStr,FixFileLine,ParamFileLine: String;
  LaserOff,ShowStuff: Boolean;
  QSize: LongInt;
  i,IOerr: Integer;
  LaserPower: LaserPowerType;

{
   Cursor on / off routines
}

function HexByte(num:Byte):str4;
const
```

```
    HexChars: str16= '0123456789ABCDEF';
begin
    HexByte := HexChars[num shr 4 + 1] + HexChars[num and
     $0F +1];
end;

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;
```

```
{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln('ERROR: '+Text);
  beep;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'POWER: ',Text);
  close(ErrorFile);
end;

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
begin
  if whereX>1 then writeln;
  writeln('*** ',Text);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
   upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
```

```
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = CtrlC then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
```

```
end;

procedure DrawBox(x1,y1,x2,y2: integer;BoxType: integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                    {  Type          1    2    3    4     }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
   if   x1<x2   then   FullHorzBar   :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
```

```
  gotoxy(x1,y2);
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;

{$i lctrl.inc}

procedure ShowPowerStatus;
begin
  gotoxy(12,24);
  case LaserPower of
    _Unknown: write('U');
    _Level  : write('L');
    _Toggle : write('T');
  end;
  write('-Laser ');
  if port[PowerPort + 2] and _LASER_IN = 0 then
write('On ')
    else write('Off');
  write('  Shutter ');
  if port[PowerPort + 2] and _SHUTTER_IN = 0 then
write('Open  ')
    else write('Closed');
  write('  Door ');
  if port[PowerPort + 2] and _DOOR_IN = 0 then
write('Open  ')
    else write('Closed');
  write('  Elevator ');
  if port[PowerPort + 2] and _ZSTAGE_OUT = 0 then
write('Off')
    else write('On ');
end;

procedure SendLaserCommand(cmd: String);
begin
  ShowError(STEREO(cmd));
```

```
end;

procedure PowerControl(bit: Integer;Op: OnOffType);
begin
  if Op = On then
    port[PowerPort + 2] := port[PowerPort + 2] or bit
  else
    port[PowerPort + 2] := port[PowerPort + 2] and
($FF-bit);
end;

procedure ToggleLaserPowerOn;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN = 0) or (loop1
> 100000);
  PowerControl(_LASER_OUT,Off);
  if (loop1 > 100000) then
    PrintError('Can''t detect laser toggle feedback.');
end;

procedure ToggleLaserPowerOff;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN > 0) or (loop1
> 100000);
  PowerControl(_LASER_OUT,Off);
```

```
  if (loop1 > 100000) then
    PrintError('Can''t detect laser toggle feedback.');
end;

procedure TurnOnLaser;
var
  loop: LongInt;
begin
  if LaserPower = _Toggle then begin
    if port[PowerPort + 2] and _LASER_IN > 0 then
ToggleLaserPowerOn;
  end else PowerControl(_LASER_OUT,On);
  if LaserPresent and StereoPresent then begin
    SendLaserCommand('HR;CL;MD BL +');
    assign(FixFile,FixFileName);
    {$I-}
    reset(FixFile);
    IOerr := IOresult;
    if IOerr <> 0 then begin
      assign(FixFile,HomeDirectory + FixFileName);
      reset(FixFile);
      IOerr := IOresult;
    end;
    {$I+}
    if IOerr = 0 then begin
      while not eof(FixFile) do begin
        readln(FixFile,FixFileLine);
        if (FixFileLine <> '') and (FixFileLine[1] <> '!')
then
          SendLaserCommand(FixFileLine);
      end;
      close(FixFile);
    end;
  end;
end;

procedure TurnOnElevator;
```

```
begin
  PowerControl(_ZSTAGE_OUT,On);
end;

procedure OpenShutter;
begin
  PowerControl(_SHUTTER_OUT,On);
end;

procedure TurnOffLaser;
begin
  gotoxy(18,17);
  write('--- WARNING ---');
  gotoxy(18,18);
  write('The laser may take half an hour to restart!');
  gotoxy(18,19);
  write('Are you sure you want to turn the laser off? ');
  repeat
    ShowPowerStatus;
  until keypressed;
  HandleKeyboardRequest;
  if upcase(key)='Y' then
    if LaserPower = _Toggle then begin
      if port[PowerPort + 2] and _LASER_IN = 0 then
ToggleLaserPowerOff;
    end else PowerControl(_LASER_OUT,Off);
end;

procedure TurnOffElevator;
begin
  PowerControl(_ZSTAGE_OUT,Off);
end;

procedure CloseShutter;
begin
  PowerControl(_SHUTTER_OUT,Off);
end;
```

```
procedure GetSelection;
begin
  repeat
    repeat
      ShowPowerStatus;
    until keypressed;
    HandleKeyboardRequest;
    key := upcase(key);
    if key in ['1'..'6','X'] then exit;
  until false;
end;

procedure ShowPowerMenu;
begin
  window(1,1,80,21);
  ClrScr;
  centerln('POWER, SLA-1 Power Control Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  for i:=1 to 5 do writeln;
  DrawBox(Indent-1,8,80-Indent,16,1);
  window(Indent,9,80-Indent-1,15);
  ClrScr;
  writeln(' 1. Turn Laser and Mirror Drive On');
  writeln(' 2. Turn Elevator On');
  writeln(' 3. Open Shutter');
  writeln(' 4. Turn Laser and Mirror Drive Off');
  writeln(' 5. Turn Elevator Off');
  writeln(' 6. Close Shutter');
  write(' X. Exit');
  window(1,1,80,25);
  gotoxy(28,19);
  write('Please make a selection.');
  DrawBox(10,23,69,25,1);
end;
```

```
procedure HardReset;
begin
  port[PowerPort + 3] := $98;    { init 8255 chip }
  port[PowerPort + 3] := $98;
end;

procedure InitPowerPort;
begin
  PowerControl(1,Off);
  if port[PowerPort + 2] and _INIT_IN > 0 then HardReset
  else begin
    PowerControl(1,On);
    if port[PowerPort + 2] and _INIT_IN = 0 then
HardReset;
  end;
end;

procedure ReadParams;
begin
  assign(ParamFile,ParamFileName);
  {$I-}
  reset(ParamFile);
  IOerr := IOresult;
  if IOerr <> 0 then begin
    assign(ParamFile,HomeDirectory + ParamFileName);
    reset(ParamFile);
    IOerr := IOresult;
  end;
  {$I+}
  LaserPower := _Unknown;
  if IOerr = 0 then begin
    while not eof(ParamFile) do begin
      readln(ParamFile,ParamFileLine);
      ParamFileLine := upper(ParamFileLine);
      if (ParamFileLine <> '') and (ParamFileLine[1] <>
'!') then begin
```

```
        if copy(ParamFileLine,1,15) = 'LASERPOWERTYPE='
then begin
          temp := copy(ParamFileLine,16,3);
          if temp = 'LEV' then LaserPower := _Level
          else if temp = 'TOG' then LaserPower := _Toggle;
        end;
      end;
    end;
    close(ParamFile);
  end;
end;
begin
  CheckBreak := false;
  ShowStuff  := false;
  LaserOff   := false;
  ClrScr;
  InitPowerPort;
  IndentStr := many(' ',Indent);
  ReadParams;
  CursorOff;
  repeat
    ShowPowerMenu;
    GetSelection;
    gotoxy(1,22);
    ClrEol;
    case key of
      '1': TurnOnLaser;
      '2': TurnOnElevator;
      '3': OpenShutter;
      '4': TurnOffLaser;
      '5': TurnOffElevator;
      '6': CloseShutter;
      'X': Abort;
    end;
  until false;
end.
^Z
```

```
'=========================================================
'  3D Systems Stereolithography System Software
'=========================================================
'
'  PREPARE.BAS
'  SLA-1 Prepare for Part-Making Software
'
'---------------------------------------------------------
'
'  History:
'  11/12/87  Ver 2.30  first Beta software release
'  11/13/87  Ver 2.31  new material data input fix
'                      range manager and CalcSp fixes
'  12/02/87  Ver 2.32  removed Material Input capability
'  12/04/87  Ver 2.33  programmable block updating
'                  BLOCK.ID file support
'  12/14/87  Ver 2.40  Material linewidths (min & max)
'                      support
'  12/17/87  Ver 2.41  no last material file condition
'                      handled
'  12/22/87  Ver 2.43  no blanking scan before report
'   1/06/88  Ver 2.50  update depth when changing step
'                      period value no overflow on step
'                      period input
'   1/11/88  Ver 2.51  BLOCK.ID looked for in \3DSYS too
'   1/20/88  Ver 2.54  paging occurs for more than 10
'                      blocks support of new *.R Range
'                      Control File added /SCAN option
'                      to force block scan
'   1/25/88  Ver 2.60  Calc SP placed under Range
'                      Manager new Vector Manager (old
'                      Layer Edit) Range File and Layer
'                      File command update separated out
'                      PREPARE.PRM support different SS
'                      (Step Size) support for skins
'                      part file statistics
```

```
'   2/03/88  -          old range files identified
'                       operator may proceed to build new
'                       range file
'   2/08/88  -          different layer block cmds bug
'                       removed
'   2/12/88  Ver 2.61   Comments array indexing starts at
'                       1 not 0 added Block Scan will '
wipe out commands warning query
'   2/26/88  Ver 2.62   default UpdateCmds group is not 1
'                       instead of 0
'   3/09/88  Ver 2.80   added copyright notice
'   4/04/88  Ver 3.00   menu changes
'   4/06/88  Ver 3.01   Parameter Manager code installed
'                       removed *.V file editing
'                       capability
'   4/08/88  Ver 3.02   SUPER renamed as BUILD
'   4/11/88  Ver 3.03   allow first and last layers to be
'                       deleted if other ranges exist
'   4/12/88  Ver 3.04   additional prompts to Parameter
'                       Manager correct open file error
'                       in Parameter Manager
'   4/15/88  Ver 3.05   store last part file name in
'                       PART.LST file
'-----------------------------------------------------

  DIM
ZStart&[100],ZEnd&[100],ZSpace%[100],CureDepth%[100]
  DIM BlockStart%[100],BlockEnd%[100]
  DIM BlockType$[3000],BlockCmd$[3000]
  DIM TBType$[30],TBCmd$[30]
  DIM SP[20],LH[20],WMIN[20],WMAX[20]
  DIM BlockGroupCount%[10],BlockGroupId$[10,40]
  DIM Comments$[100]
  DIM Offsets&[50,2]

  MaxInt = 32768
  MaxUInt = 65535
```

```
  TRUE = -1
  FALSE = 0
  ScanFlag% = FALSE
  VersionId$ = "3.05"
  Title$ = "Prepare V" + VersionId$
  HomeDir$ = "C:\3DSYS\"
  ParamFileName$      = "PREPARE.PRM"
  BuildParamFileName$ = "BUILD.PRM"
  TempParamFileName$  = "TEMP.PRM"
  LastPartFileName$   = "PART.LST"

  ON ERROR GOTO TryHome
  OPEN ParamFileName$ FOR INPUT AS #1
  GOTO ReadParams
TryHome:
  RESUME TryHome2
TryHome2:
  ON ERROR GOTO NoParams
  OPEN HomeDir$ + ParamFileName$ FOR INPUT AS #1
  GOTO ReadParams
NoParams:
  NormStepSize$ = "SS 2"
  FillStepSize$ = "SS 16"
  RESUME PrepareInit
'
' read PREPARE's parameter file
'
ReadParams:
  LINE INPUT #1,A$
  IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN ReadParams
  I% = INSTR(2,A$,CHR$(34))
  IF I% > 0 THEN A$ = LEFT$(A$,I%)
  CmdLine$ = FN NoQuotes$(A$)
  GOSUB CheckForOptions
  FOR I%=1 TO 2  ' change this when altering PREPARE.PRM
format
ReadNextLine:
```

```
    LINE INPUT #1,A$
    IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN ReadNextLine
    V=VAL(A$)
    V$ = LEFT$(A$,1)
    IF V$=CHR$(34) THEN
      V% = INSTR(MID$(A$,2),CHR$(34)) + 1
      S$ = MID$(A$,2,V%-2)
    ELSE
      S$ = ""
    END IF
    IF      I%=1 THEN NormStepSize$ = S$_
    ELSE IF I%=2 THEN FillStepSize$ = S$
  NEXT I%
  CLOSE #1

PrepareInit:
  SingleBar$ = STRING$(45,196)
  RangeBar$ = ""
  FOR I%=1 TO 8
    READ S%
    IF I%>1 THEN RangeBar$ = RangeBar$ + CHR$(249)
    RangeBar$ = RangeBar$ + STRING$(S%,196)
  NEXT I%
  DATA 7,9,7,9,11,11,11,7
  BlockBar$ = ""
  FOR I%=1 TO 3
    READ S%
    IF I%>1 THEN BlockBar$ = BlockBar$ + CHR$(249)
    BlockBar$ = BlockBar$ + STRING$(S%,196)
  NEXT I%
  DATA 8,10,50
  LayerBar$ = ""
  FOR I%=1 TO 5
    READ S%
    IF I%>1 THEN LayerBar$ = LayerBar$ + CHR$(249)
    LayerBar$ = LayerBar$ + STRING$(S%,196)
  NEXT I%
```

```
  DATA 7,9,7,9,7
  SpBar$ = "
  FOR I%=1 TO 8
    READ S%
    IF I%>1 THEN SpBar$ = SpBar$ + CHR$(249)
    SpBar$ = SpBar$ + STRING$(S%,196)
  NEXT I%
  DATA 7,9,7,9,7,8,7,7
  PowerFile$          = "POWER.LST"
  LastMaterialFile$   = "MATERIAL.LST"
  BlockIdFileName$    = "BLOCK.ID"
  MinSp& = 8
  MaxSp& = 4000

  COLOR 15
Restart:
  CLS
  PRINT SingleBar$
  PRINT "  PREPARE, Part-Making Preparer"
  PRINT "  Version ";VersionId$
  PRINT "  Copyright (C) 1988 by 3D Systems, Inc."
  PRINT "  3D Systems Laser Stereolithography System"
  PRINT SingleBar$
  PRINT
  PRINT

  BlockGroups% = 0
  ON ERROR GOTO TryHome3
  OPEN BlockIdFileName$ FOR INPUT AS #1
  GOTO ReadBlockId
TryHome3:
  ON ERROR GOTO NoBlockIdFile
  OPEN "C:\3DSYS\"+BlockIdFileName$ FOR INPUT AS #1
  RESUME ReadBlockId
ReadBlockId:
  ON ERROR GOTO 0
  IF EOF(1) THEN ProcessCmdLine
```

```
    LINE INPUT #1,A$
    IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN ReadBlockId
    Index% = 1
    CALL UnitValShortInt(Group%,A$,Index%)
    CALL UnitStr(Block$,A$,Index%)
    IF Group%<0 OR Group%>10 THEN
      CALL SystemError("Invalid block group number in
Block Id File "_
        +BlockIdFileName$)
      END
    END IF
    IF Group% > BlockGroups% THEN BlockGroups% = Group%
    BlockGroupId$[Group%,BlockGroupCount%[Group%]] =
UCASE$(Block$)
    INCR BlockGroupCount%[Group%]
    GOTO ReadBlockId

ProcessCmdLine:
    CLOSE #1
    CmdLine$ = UCASE$(Command$)
    GOSUB CheckForOptions
    IF LEN(CmdLine$)=0 THEN
      PRINT "Part files in this directory:"
      ON ERROR GOTO NoPartFiles
      FILES "*.V"
      GOTO AskForPartFile
NoPartFiles:
      PRINT "- none -"
      RESUME AskForPartFile
AskForPartFile:
      PRINT
      INPUT "Part Name: ",CmdLine$
      PRINT
    END IF
    CmdLine$ = UCASE$(CmdLine$)
    GOSUB CheckForOptions
    I% = INSTR(CmdLine$,".")
```

```
  IF I%>0 THEN
    PartName$ = LEFT$(CmdLine$,I%-1)
  ELSE
    PartName$ = CmdLine$
  END IF
  IF LEN(PartName$)=0 THEN END
  OPEN LastPartFileName$ FOR OUTPUT AS #1
  PRINT #1,PartName$
  CLOSE #1
  GOTO StartPrepare

CheckForOptions:
  I% = INSTR(CmdLine$,"/SCAN")
  IF I%>0 THEN
    ForcedScan% = TRUE
    CmdLine$ = LEFT$(CmdLine$,I%-1)+MID$(CmdLine$,I%+5)
  END IF
  RETURN

'-----------------------------------------------------

StartPrepare:
  PartRangeFileName$  = PartName$ + ".R"
  PartLayerFileName$  = PartName$ + ".L"
  BackupFileName$ = PartName$ + ".BAK"
  ON ERROR GOTO NewPart
  OPEN PartRangeFileName$ FOR INPUT AS #10
  ON ERROR GOTO 0
  I% = 0
CheckOld:
  IF EOF(10) THEN HaveOldRangeFile
  LINE INPUT #10,L$
  L1$ = UCASE$(LEFT$(L$,1))
  IF L1$="" OR L1$="!" OR L1$="*" THEN CheckOld
  IF L1$ >= "A" AND L1$ <= "Z" THEN HaveNewRangeFile
  INCR I%
  IF I% < 30 THEN CheckOld
```

```
HaveOldRangeFile:
  PRINT
  PRINT "File ";PartRangeFileName$;" is an older Range
File with a";
  PRINT " different format."
  PRINT "Do you wish to continue and build a new Range
File from the Layer"
  PRINT "File ";PartLayerFileName$;"? ";
  INPUT "",Q$
  IF UCASE$(LEFT$(Q$,1))="Y" THEN
    PRINT
    GOTO BuildRangeTable
  ELSE
    STOP
  END IF

HaveNewRangeFile:
  CLOSE #10
  OPEN PartRangeFileName$ FOR INPUT AS #10
  I% = 0
  GotBlocksFromRangeFile% = FALSE
  CommentCount% = 0
RangeRead:
  ON ERROR GOTO 0
  LINE INPUT #10,L$
  R$=UCASE$(LEFT$(L$,1))
  IF R$="*" THEN
    Index% = 2
    WHILE MID$(L$,Index%,1)=" "
      Index% = Index% + 1
    WEND
    CALL UnitVal(SliceRes,L$,Index%)
    CALL UnitVal(ZSpacing,L$,Index%)
    CALL UnitVal(Thickness,L$,Index%)
    GOTO RangeRead
  END IF
  IF LEN(L$)=0 OR R$="!" THEN
```

```
    IF CommentCount% <= 100 THEN
      INCR CommentCount%
      Comments$[CommentCount%] = L$
    END IF
    GOTO RangeRead
  END IF
  RangeIndex% = 1
  IF R$>="0" AND R$<="9" THEN
    INCR I%
    CALL UnitValInt(ZStart&[I%],L$,RangeIndex%)
    CALL UnitValInt(ZEnd&  [I%],L$,RangeIndex%)
    CALL UnitValShortInt(ZSpace%[I%],   L$,RangeIndex%)
    CALL UnitValShortInt(CureDepth%[I%],L$,RangeIndex%)
    BlockStart%[I%] = BlockEnd%
  ELSE
    GotBlocksFromRangeFile% = TRUE
    CALL UnitStr(BlockType$[BlockEnd%],L$,RangeIndex%)
    T$ = FN NoSp$(MID$(L$,RangeIndex%))
    J%=INSTR(T$,"!")
    IF J%=0 THEN J%=LEN(T$)+1
    BlockCmd$[BlockEnd%] = FN NoSp$(LEFT$(T$,J%-1))
    BlockEnd%[I%] = BlockEnd%
    INCR BlockEnd%
  END IF
  IF NOT EOF(10) THEN RangeRead
  RangeCount% = I%
  CLOSE #10
'  backwards (pre V2.55) compatibility
  IF ForcedScan% OR NOT GotBlocksFromRangeFile% THEN
GOSUB BlockScan
  GOTO MainMenu

NewPart:
  RESUME BuildRangeTable

BuildRangeTable:
  ON ERROR GOTO NoLayerFile
```

```
  OPEN PartLayerFileName$ FOR INPUT AS #3
  ON ERROR GOTO 0
  PRINT "Reading ";PartLayerFileName$;"..."

GetToControlTop:
  IF EOF(3) THEN
    CALL SystemError("Error:  "+PartLayerFileName$+"
empty.")
    END
  END IF
  LINE INPUT #3,CONTROL$
  C$=LEFT$(CONTROL$,1)
  IF C$="!" THEN GetToControlTop
  IF C$="*" THEN
    Index% = 2
    WHILE MID$(CONTROL$,Index%,1)=" "
      Index% = Index% + 1
    WEND
    CALL UnitVal(SliceRes,CONTROL$,Index%)
    CALL UnitVal(ZSpacing,CONTROL$,Index%)
    CALL UnitVal(Thickness,CONTROL$,Index%)
    GOTO GetToControlTop
  END IF

  RangeCount% = 0
  BlockIndex% = 0
  TBIndex% = 0
  First% = TRUE
  Last%  = FALSE
  GOTO AlreadyHaveBlock

ReadBlocks:
  IF EOF(3) THEN
    Last% = TRUE
    GOTO CompareLayer
  END IF
  LINE INPUT #3,CONTROL$
```

```
AlreadyHaveBlock:
  IF LEN(CONTROL$)=0 OR LEFT$(CONTROL$,1)="!" THEN
ReadBlocks
'
'  parse out layer number, name, command, and comment
'
  ControlIndex% = 1
  CALL UnitValInt(ControlLayer&,CONTROL$,ControlIndex%)
  CALL UnitStr(ControlType$,CONTROL$,ControlIndex%)
  T$ = FN NoSp$(MID$(CONTROL$,ControlIndex%))
  I%=INSTR(T$,"!")
  IF I%=0 THEN I%=LEN(T$)+1
  ControlCmd$ = FN NoSp$(LEFT$(T$,I%-1))

'
'  store blocks in temporary array, loop until different
layer
'
StoreBlocks:
  IF ControlType$="#CA" THEN ReadBlocks    ' skip
critical areas
  IF First% THEN    ' handle first layer
    First% = FALSE
    RangeStep% = ZSpacing
    Layer& = ControlLayer&
    GOSUB StartNewRange
    GOTO StoreBlock
  END IF
  IF ControlLayer& <> Layer& THEN EvaluateLayer
StoreBlock:
  TBType$[TBIndex%] = ControlType$
  TBCmd$[TBIndex%] = ControlCmd$
  INCR TBIndex%
  GOTO ReadBlocks

'
'  start a new layer range if the Z-spacing changes
```

```
'
EvaluateLayer:
  IF NeedStep% THEN
    NeedStep% = FALSE
    RangeStep% = ControlLayer& - Layer&
    ZSpace%[RangeCount%] = RangeStep%
  END IF
  TwoLayersBack& = OldLayer&
  OldLayer& = Layer&
  Layer& = ControlLayer&
  CD% = ControlLayer& - OldLayer&
  IF CD%<>RangeStep% THEN
    GOSUB StartNewRange
    GOTO UpdateRange
  END IF

compare current layer's block commands against those of
current range if there are two different commands for a
particular layer, a new layer range must be started
'
CompareLayer:
  FOR I%=0 TO TBIndex%-1  ' check each block of current
layer
    FOR J%=BlockStart%[RangeCount%] TO
BlockEnd%[RangeCount%]
      IF TBType$[I%]=BlockType$[J%] THEN
        IF TBCmd$[I%]<>BlockCmd$[J%] THEN
          GOSUB StartNewRange
          ZEnd&  [RangeCount%-1] = TwoLayersBack&
          ZStart&[RangeCount%]   = OldLayer&
          ZEnd&  [RangeCount%]   = OldLayer&
          GOTO UpdateRange
        ELSE
          GOTO NextCurrentBlock
        END IF
      END IF
    NEXT J%
```

```
NextCurrentBlock:
  NEXT I%
'
'  layer checks out, add it to current range
'
UpdateRange:
  ZEnd&[RangeCount%] = ControlLayer&
'
'  add any new blocks to current range's block table
'
  FOR I%=0 TO TBIndex%-1  ' check each block of current
layer
    FOR J%=BlockStart%[RangeCount%] TO
BlockEnd%[RangeCount%]
      IF TBType$[I%]=BlockType$[J%] THEN
NextCurrentBlock2
    NEXT J%
    BlockType$[BlockIndex%] = TBType$[I%]
    BlockCmd$[BlockIndex%] = TBCmd$[I%]
    INCR BlockIndex%
NextCurrentBlock2:
  NEXT I%
  BlockEnd%[RangeCount%] = BlockIndex%-1
'
'  set up the new layer
'
  TBIndex% = 0
  IF Last% THEN LayerCleanUp
  GOTO StoreBlock

'
'  start a new range of layers
'
StartNewRange:
  NeedStep% = TRUE
  INCR RangeCount%
  ZStart&    [RangeCount%] = ControlLayer&
```

```
  ZEnd&       [RangeCount%] = ControlLayer&
  RangeStep% = ControlLayer& - Layer&
  BlockStart%[RangeCount%] = BlockIndex%
  BlockEnd%  [RangeCount%] = BlockIndex%
  TBIndex% = 0
  RETURN

LayerCleanUp:
  CLOSE #3
  FOR I%=1 TO RangeCount%
    CureDepth%[I%] = ZSpace%[I%]
  NEXT I%
  GOTO MainMenu

NoBlockIdFile:
  CALL SystemError("Can't open Block Id File
"+BlockIdFileName$)
  END

BadBlockIdFile:
  CALL SystemError("Bad format for Block Id File
"+BlockIdFileName$)
  END

NoLayerFile:
  CALL SystemError("Can't open Layer Control File
"+PartLayerFileName$)
  END

NoRangeFile:
  CALL SystemError("Can't open Range File
"+PartRangeFileName$)
  END

RangeFileProb:
  CALL SystemError("Problem with Range File
"+PartRangeFileName$)
```

```
  END

NoBackupFile:
  CALL SystemError("Can't open Backup File
"+BackupFileName$)
  END

NoBuildParamFile:
  CALL SystemError("Can't find BUILD Parameter File
"+BuildParamFileName$)
  END


'-------------------------------------------------------
'  show PREPARE's Main Menu
'
MainMenu:
  CLS
  I$ = STRING$(29," ")
  MainBar$=STRING$(25,CHR$(196))
  T$ = "MAIN MENU"
  GOSUB CenterLine
  T$ = Title$
  GOSUB CenterLine
  T$ = "working on part "+PartName$
  GOSUB CenterLine
  PRINT
  PRINT
  T$ = MainBar$
  GOSUB CenterLine
  PRINT
  PRINT
  PRINT I$;"1. Parameters Manager"
  PRINT I$;"2. Range Manager"
  PRINT I$;"3. Layer Manager"
  PRINT I$;"4. Statistics"
  PRINT I$;"X. Exit"
```

```
  PRINT
  PRINT
  T$ = MainBar$
  GOSUB CenterLine
  PRINT
  PRINT
  PRINT I$;"Which option? ";
LoopMain:
  C$ = UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopMain
  IF C$="X" THEN END
  IF C$="1" THEN ParamsManager
  IF C$="2" THEN RangeManager
  IF C$="3" THEN LayerManager
  IF C$="4" THEN Statistics
  GOTO LoopMain


'=============================================================
'
'  Parameters Manager:
'
'
ParamsManager:
  ON ERROR GOTO BuildParamsNotThisDir
  OPEN BuildParamFileName$ FOR INPUT AS #2
  F$ = BuildParamFileName$
  GOTO ReadBuildParamsFile

BuildParamsNotThisDir:
  RESUME TryHome4

TryHome4:
  ON ERROR GOTO NoBuildParamFile
  OPEN HomeDir$ + BuildParamFileName$ FOR INPUT AS #2
  F$ = HomeDir$ + BuildParamFileName$
```

```
' read Builder parameter file
'
ReadBuildParamsFile:
  LINE INPUT #2,A$
  IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN
ReadBuildParamsFile
  BuildCmdLine$ = FN NoQuotes$(A$)
  FOR I%=1 TO 8  ' change this when altering BUILD.PRM
format
NextGenLine:
    LINE INPUT #2,A$
    IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN NextGenLine
    V=VAL(A$)
    IF      I%=1 THEN ZAddress%       = V_
    ELSE IF I%=2 THEN ZPitch          = V_
    ELSE IF I%=3 THEN XYScale         = V_
    ELSE IF I%=4 THEN GOSUB GetXOffsets  _
    ELSE IF I%=5 THEN GOSUB GetYOffsets  _
    ELSE IF I%=6 THEN MaxVectorCount& = V_
    ELSE IF I%=7 THEN GOSUB GetLowerViewportXY_
    ELSE IF I%=8 THEN GOSUB GetUpperViewportXY
  NEXT I%
  NumOffsets% = NumXOffsets%
  IF NumOffsets% > NumYOffsets% THEN NumOffsets% =
NumYOffsets%
  CLOSE #2
  GOTO ShowBuildParams

GetXOffsets:
  NumXOffsets% = 0
GetMoreXOffsets:
  Offsets&[NumXOffsets%,1] = VAL(A$)
  INCR NumXOffsets%
  J% = FN FindStr(1,A$,", ")
  IF J%=LEN(A$)+1 THEN XOffsetRet
  IF MID$(A$,J%,1)=" " THEN XOffsetRet
  A$ = MID$(A$,J%+1,199)
```

```
  IF NumXOffsets%<50 THEN GetMoreXOffsets
XOffsetRet:
  RETURN

GetYOffsets:
  NumYOffsets% = 0
GetMoreYOffsets:
  Offsets&[NumYOffsets%,2] = VAL(A$)
  INCR NumYOffsets%
  J% = FN FindStr(1,A$,", ")
  IF J%=LEN(A$)+1 THEN YOffsetRet
  IF MID$(A$,J%,1)=" " THEN YOffsetRet
  A$ = MID$(A$,J%+1,199)
  IF NumYOffsets%<50 THEN GetMoreYOffsets
YOffsetRet:
  RETURN

GetLowerViewportXY:
  LowerViewPortX& = VAL(A$)
  J% = FN FindStr(1,A$,", ")
  LowerViewPortY& = VAL(MID$(A$,J%+1,99))
  RETURN

GetUpperViewportXY:
  UpperViewPortX& = VAL(A$)
  J% = FN FindStr(1,A$,", ")
  UpperViewPortY& = VAL(MID$(A$,J%+1,99))
  RETURN

ShowBuildParams:
  CLS
  T$ = "PARAMETERS MANAGER"
  GOSUB CenterLine
  T$ = Title$
  GOSUB CenterLine
  T$ = "working on part "+PartName$
  GOSUB CenterLine
```

```
  PRINT
  T$ = SingleBar$
  GOSUB CenterLine
  PRINT
  Indent$ = STRING$(2," ")
  PRINT Indent$;"1. Z-Pitch. . . . . .";ZPitch
  PRINT Indent$;"2. XY-Only Scale. . . . . .";INT
     (XYScale*10000)/10000
  PRINT Indent$;"3. Maximum Vector Count . . . . . .";
     MaxVectorCount&
  PRINT Indent$;"4. Minimum Viewport Coordinates . . ";_
    USING "(#####";LowerViewportX&;
  PRINT ",";USING "#####)";LowerViewportY&
  PRINT Indent$;"5. Maximum Viewport Coordinates . . ";_
    USING "(#####";UpperViewportX&;
  PRINT ",";USING "#####)";UpperViewportY&
  PRINT Indent$;"6. Multiple Part Positioning. . . .";
  FOR I%=0 TO NumOffsets% - 1
    IF I% > 0 AND I% MOD 3 = 0 THEN PRINT : PRINT
Indent$;STRING$(35," ");
    PRINT USING " (#####";Offsets&[I%,1];
    PRINT ",";USING "#####)";Offsets&[I%,2];
  NEXT I%
  PRINT
  PRINT Indent$;"7. BUILD Options Line:"
  PRINT Indent$;"   ";BuildCmdLine$
  PRINT Indent$;"U. Update BUILD Parameters File on
disk"
  PRINT Indent$;"X. Exit"
  PRINT
  T$ = SingleBar$
  GOSUB CenterLine
  PRINT
LoopParams:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopParams
  IF C$="1" THEN EditZPitch
```

```
  IF C$="2" THEN EditXYScale
  IF C$="3" THEN EditMaxVectorCount
  IF C$="4" THEN EditLowerViewport
  IF C$="5" THEN EditUpperViewPort
  IF C$="6" THEN EditMultiPartPos
  IF C$="7" THEN EditBuildCmdLine
  IF C$="U" THEN UpdateBuildParamsFile
  IF C$="X" THEN MainMenu
  GOTO LoopParams

EditZPitch:
  INPUT "Enter new Z-Pitch value: ",A$
  IF LEN(A$)>0 THEN ZPitch = VAL(A$)
  GOTO ShowBuildParams

EditXYScale:
  INPUT "Enter new XY-Only Scale Factor: ",A$
  IF LEN(A$)>0 THEN XYScale = VAL(A$)
  GOTO ShowBuildParams

EditMaxVectorCount:
  INPUT "Enter new Maximum Vector Count: ",A$
  IF LEN(A$)>0 THEN MaxVectorCount& = VAL(A$)
  IF MaxVectorCount& < 1000  THEN MaxVectorCount& = 1000
  IF MaxVectorCount& > 60000 THEN MaxVectorCount& =
60000
  GOTO ShowBuildParams

EditLowerViewport:
  LINE INPUT "Enter new Lower Viewport coordinates
(x,y): ",A$
  IF LEN(A$)>0 THEN GOSUB GetLowerViewportXY
  GOTO ShowBuildParams

EditUpperViewPort:
  LINE INPUT "Enter new Upper Viewport coordinates
(x,y): ",A$
```

```
  IF LEN(A$)>0 THEN GOSUB GetUpperViewportXY
  GOTO ShowBuildParams

EditMultiPartPos:
  PRINT
  PRINT " If Position exists, it will be deleted."
  PRINT " If Position does not exist, it will be
added.";
  LOCATE CSRLIN - 2 , 1
  LINE INPUT "Enter Position to add or remove (x,y):
",A$
  IF LEN(A$)>0 THEN
    X& = VAL(A$)
    J% = FN FindStr(1,A$,", ")
    Y& = VAL(MID$(A$,J%+1,99))
    ' if (X,Y) already present, delete them from list
    FOR I%=0 TO NumOffsets%-1
      IF Offsets&[I%,1] = X& AND Offsets&[I%,2] = Y&
THEN
        FOR J%=I% TO NumOffsets%-2
          Offsets&[J%,1] = Offsets&[J%+1,1]
          Offsets&[J%,2] = Offsets&[J%+1,2]
        NEXT J%
        DECR NumOffsets%
        GOTO ShowBuildParams
      END IF
    NEXT I%
    ' if (X,Y) not present, add them to list
    IF NumOffsets% < 50 THEN
      Offsets&[NumOffsets%,1] = X&
      Offsets&[NumOffsets%,2] = Y&
      INCR NumOffsets%
    END IF
  END IF
  GOTO ShowBuildParams

EditBuildCmdLine:
```

```
  PRINT
  PRINT "(Enter a Space to clear)";
  LOCATE CSRLIN - 1 , 1
  LINE INPUT "Enter new BUILD Options: ",A$
  IF LEN(A$)>0 THEN BuildCmdLine$ = A$
  GOTO ShowBuildParams

UpdateBuildParamsFile:
  OPEN F$ FOR INPUT AS #2
  OPEN TempParamFileName$ FOR OUTPUT AS #1
GetBuildParamsFileLine:
  LINE INPUT #2,A$
  IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN
    PRINT #1,A$
    GOTO GetBuildParamsFileLine
  END IF
  PRINT #1,CHR$(34);BuildCmdLine$;CHR$(34)    ' surround
with quotes
  FOR I%=1 TO 8    ' change this when altering BUILD.PRM
format
NextBuildParamLine:
    LINE INPUT #2,A$
    IF LEN(A$)=0 OR LEFT$(A$,1)="!" THEN
      PRINT #1,A$
      GOTO NextBuildParamLine
    END IF
    Index% = 1
ScanForNonNumeric:
    C$ = MID$(A$,Index%,1)
    IF (C$ >= "0" AND C$ <= "9") OR C$ = "." OR C$=","
THEN
      INCR Index%
      GOTO ScanForNonNumeric
    END IF
    Comment$ = MID$(A$,Index%)
    IF      I%=1 THEN PRINT #1,FN
NoSp$(STR$(ZAddress%));_
```

```
    ELSE IF I%=2 THEN PRINT #1,FN
NoSp$(STR$(INT(ZPitch*10000)/10000));_
    ELSE IF I%=3 THEN PRINT #1,FN
NoSp$(STR$(INT(XYScale*10000)/10000));_
    ELSE IF I%=4 THEN GOSUB PrintXOffsets_
    ELSE IF I%=5 THEN GOSUB PrintYOffsets_
    ELSE IF I%=6 THEN PRINT #1,FN
NoSp$(STR$(MaxVectorCount&));_
    ELSE IF I%=7 THEN GOSUB PrintLowerViewportXY_
    ELSE IF I%=8 THEN GOSUB PrintUpperViewportXY
    PRINT #1,Comment$
  NEXT I%
  WHILE NOT EOF(2)
    LINE INPUT #2,A$
    PRINT #1,A$
  WEND
  CLOSE #1
  CLOSE #2
  ' replace BUILD.PRM in current directory
  ' or make BUILD.PRM from \3DSYS\BUILD.PRM Master
  IF F$ = BuildParamFileName$ THEN KILL F$
  NAME TempParamFileName$ AS BuildParamFileName$
  GOTO ShowBuildParams

PrintXOffsets:
  FOR J%=0 TO NumOffsets%-1
    IF J% > 0 THEN PRINT #1,",";
    PRINT #1,FN NoSp$(STR$(Offsets&[J%,1]));
  NEXT J%
  RETURN

PrintYOffsets:
  FOR J%=0 TO NumOffsets%-1
    IF J% > 0 THEN PRINT #1,",";
    PRINT #1,FN NoSp$(STR$(Offsets&[J%,2]));
  NEXT J%
  RETURN
```

```
PrintLowerViewportXY:
  PRINT #1,FN NoSp$(STR$(LowerViewportX&));",";_
    FN NoSp$(STR$(LowerViewportY&));
  RETURN

PrintUpperViewportXY:
  PRINT #1,FN NoSp$(STR$(UpperViewportX&));",";_
    FN NoSp$(STR$(UpperViewportY&));
  RETURN


'=========================================================='
'
'  Range Manager:
'
'  show layer range table
'  a layer range is conditioned on the following:
'    all layers must have the same z-spacing from the
last layer
'    they must have the same command strings for each
type of block
'  allow comments (stored in first layer only)
'
RangeManager:
  CLS
  IF RangeCount% = 0 THEN
    PRINT "No layer ranges!"
  ELSE
    T$ = "RANGE MANAGER"
    GOSUB CenterLine
    T$ = Title$
    GOSUB CenterLine
    T$ = "working on file "+PartRangeFileName$
    GOSUB CenterLine
    PRINT
    PRINT RangeBar$
```

```
    PRINT " Range   Z-Start   Z-End   Z-Space
CAD-Start    CAD-End    CAD-Space   Count"
    PRINT RangeBar$
    FOR I%=1 TO RangeCount%
      PRINT USING " ### ";I%;
      PRINT USING "     ##### ";ZStart&[I%];
      PRINT USING "   #####";ZEnd&[I%];
      PRINT USING "    ##### ";ZSpace%[I%];
      PRINT USING "   #####.###";ZStart&[I%]/SliceRes;
      PRINT USING "   #####.###";ZEnd&[I%]/SliceRes;
      PRINT USING "   #####.###";ZSpace%[I%]/SliceRes;
      IF ZSpace%[I%]>0 THEN
        Z% = (ZEnd&[I%]-ZStart&[I%])/ZSpace%[I%]
      ELSE
        Z% = 0
      END IF
        PRINT USING "   #####";Z%+1
    NEXT I%
    PRINT RangeBar$
  END IF
  PRINT
  PRINT "   Press:"
  PRINT "    A  add range        D  delete range
 E  edit range  "
  PRINT "    V  verify blocks    R  make a report
 C  calc SP's   "
  PRINT "    X  exit             S  save changes on disk
                "
LoopRange:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopRange
  IF C$="A" THEN AddRange
  IF C$="D" THEN DeleteRange
  IF C$="E" THEN EditRange
  IF C$="V" THEN ScanBlocks
  IF C$="R" THEN MakeReport
  IF C$="C" THEN CalcSp
```

```
  IF C$="S" THEN UpdateRangeFile
  IF C$="X" THEN MainMenu
  GOTO LoopRange

'-------------------------------------------------------

AddRange:
  PRINT
  PRINT
  PRINT "Adding Range"
  PRINT "Press Z to Enter Z-Layer or C to enter CAD
Number: ";
LoopAdd:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopAdd
  PRINT C$
  PRINT "Enter break point, in ";
  IF C$="C" THEN PRINT "CAD"; ELSE PRINT "Z-Layer";
  INPUT " units: ",RangeBreak
  IF LEN(C$)=0 THEN RangeManager
  IF C$="C" THEN
    RangeBreak& = RangeBreak*SliceRes
  ELSE
    RangeBreak& = RangeBreak
  END IF
  IF RangeBreak& <= ZStart&[1] OR RangeBreak& >
ZEnd&[RangeCount%]_
    THEN RangeManager
  FOR I%=1 TO RangeCount%
    IF RangeBreak&=ZStart&[I%] THEN RangeManager
    IF RangeBreak&<ZStart&[I%] THEN StartFound
  NEXT I%
StartFound:
  DECR I%
  ZNew& = ZStart&[I%]
  DO                ' calc new range end
    OldZNew& = ZNew&
```

```
    ZNew& = ZNew& + ZSpace%[I%]
  LOOP UNTIL ZNew&>=RangeBreak&
  IF ZNew&<>RangeBreak& THEN RangeManager  ' invalid
layer
  FOR J%=RangeCount% TO I% STEP -1
    ZStart&     [J%+1] = ZStart&     [J%]
    ZEnd&       [J%+1] = ZEnd&       [J%]
    ZSpace%     [J%+1] = ZSpace%     [J%]
    CureDepth% [J%+1] = CureDepth% [J%]
    BlockStart%[J%+1] = BlockStart%[J%]
    BlockEnd%  [J%+1] = BlockEnd%  [J%]
  NEXT J%
  INCR RangeCount%
  ZStart&[I%+1] = RangeBreak&
  ZNew& = ZStart&[I%]
  DO                ' calc new range end
    OldZNew& = ZNew&
    ZNew& = ZNew& + ZSpace%[I%]
  LOOP UNTIL ZNew&>=RangeBreak&
  ZEnd&[I%] = OldZNew&
  BlockStart%[I%] = BlockIndex%
  FOR J%=BlockStart%[I%+1] TO BlockEnd%[I%+1]
    BlockType$[BlockIndex%] = BlockType$[J%]
    BlockCmd$ [BlockIndex%] = BlockCmd$ [J%]
    INCR BlockIndex%
  NEXT J%

  BlockEnd%  [I%] = BlockIndex%-1
  GOTO RangeManager

'-------------------------------------------------------

DeleteRange:
  IF RangeCount%=1 THEN RangeManager
  PRINT
  PRINT
  INPUT "Delete what range: ",Range%
```

```
  IF Range%<1 OR Range%>RangeCount% THEN RangeManager
  IF (Range%=RangeCount%) THEN
    C$="U"
  ELSE
    C$="D"
    IF ZSpace%[Range%] = ZSpace%[Range%-1] AND _
       ZSpace%[Range%] = ZSpace%[Range%+1] THEN
      PRINT "Consolidate with range #";FN
NoSp$(STR$(Range%-1));
      PRINT " or #";FN NoSp$(STR$(Range%+1));": ";
      INPUT "",C$
      IF LEFT$(C$,1)="#" THEN C$=MID$(C$,2)
      C% = VAL(C$)
      IF C% = Range%-1 THEN C$="U"_
      ELSE IF C%=Range%+1 THEN C$="D" ELSE RangeManager
    ELSE
      IF ZSpace%[Range%] = ZSpace%[Range%-1] THEN C$="U"
ELSE C$="D"
    END IF
  END IF
  I% = Range%
  OldStart& = ZStart&[I%]
  OldEnd& = ZEnd&[I%]
  FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
    BlockType$[BlockIndex%] = "***"    ' deleted
  NEXT J%
  IF Range%<RangeCount% THEN FOR J%=I% TO RangeCount%-1
    ZStart&     [J%] = ZStart&     [J%+1]
    ZEnd&       [J%] = ZEnd&       [J%+1]
    ZSpace%     [J%] = ZSpace%     [J%+1]
    CureDepth% [J%] = CureDepth% [J%+1]
    BlockStart%[J%] = BlockStart%[J%+1]
    BlockEnd%  [J%] = BlockEnd%  [J%+1]
  NEXT J%
  DECR RangeCount%
  IF C$="D" THEN
    ZStart&[I%] = OldStart&
```

```
    ELSE
      ZEnd&[I%-1] = OldEnd&
    END IF
    GOTO RangeManager

'-----------------------------------------------------

EditRange:
    PRINT
    PRINT
    INPUT "Edit what range? ",Range%
    IF Range%<1 OR Range%>RangeCount% THEN RangeManager
    BlockBase% = BlockStart%[Range%]
    BlockCount% = BlockEnd%[Range%] - BlockStart%[Range%]
+ 1
ShowBlocks:
    CLS
    T$ = "Layer Range Blocks"
    GOSUB CenterLine
    T$ = "working on file "+PartLayerFileName$
    GOSUB CenterLine
    PRINT
    PRINT "    Range #";Range%
    PRINT "    Z-Levels:    ";
    PRINT USING "FROM      ##### TO      ##### BY
#####";_
      ZStart&[Range%],ZEnd&[Range%],ZSpace%[Range%]
    PRINT "    CAD Units:  ";
    PRINT USING "FROM #####.### TO #####.### BY
#####.###";_
      ZStart&[Range%]/SliceRes,ZEnd&[Range%]/SliceRes,_
      ZSpace%[Range%]/SliceRes
    PRINT
    PRINT "     Block#    Block Id     Block Command"
    PRINT "    ";BlockBar$
    IF BlockBase%+9 < BlockEnd%[Range%] THEN
      BlockTo% = BlockBase%+9
```

```
  ELSE
    BlockTo% = BlockEnd%[Range%]
  END IF
  FOR I%=BlockBase% TO BlockTo%
    PRINT USING "      ##   ";I%-BlockStart%[Range%]+1;
    PRINT "
";BlockType$[I%];SPC(12-LEN(BlockType$[I%]));
    PRINT BlockCmd$[I%]
  NEXT I%
  PRINT "   ";BlockBar$
  IF BlockBase%+9 < BlockEnd%[Range%] THEN PRINT "   ---
MORE BLOCKS ---"
  IF BlockBase% > BlockStart%[Range%] THEN PRINT "
Enter B to go back.";
  IF BlockBase%+9 < BlockEnd%[Range%] THEN PRINT "
Enter F to go forward.";
  PRINT
  PRINT "   Enter number of block to edit, just Return
to stop: ";
  v% = CSRLIN
  h% = POS
BlockAsk:
  LOCATE v%,h%
  PRINT "            ";
  LOCATE v%,h%
  INPUT "",Block$
  B$ = UCASE$(LEFT$(Block$,1))
  IF B$="B" THEN
    DECR BlockBase%,10
    IF BlockBase% < BlockStart%[Range%] THEN BlockBase%
= BlockStart%[Range%]
    GOTO ShowBlocks
  END IF
  IF B$="F" THEN
    IF BlockBase%+10 <= BlockEnd%[Range%] THEN INCR
BlockBase%,10
    GOTO ShowBlocks
```

```
  END IF
  IF LEN(Block$)=0 THEN RangeManager
  Block% = VAL(Block$)
  IF Block%<1 OR Block%>BlockCount% THEN
    PRINT "   Invalid block number    ";
    GOTO BlockAsk
  END IF
  Index% = BlockStart%[Range%]+Block%-1
  PRINT "   Enter commands for ";BlockType$[Index%];
  LINE INPUT ": ",Cmd$
  IF LEN(FN NoSp$(Cmd$))=0 THEN
    Cmd$ = ""
  ELSE
    IF LEFT$ (Cmd$,1)<>CHR$(34) THEN Cmd$=CHR$(34)+Cmd$
    IF RIGHT$(Cmd$,1)<>CHR$(34) THEN Cmd$=CMD$+CHR$(34)
  END IF
  BlockCmd$[Index%] = UCASE$(Cmd$)
  GOTO ShowBlocks

'-----------------------------------------------------

ScanBlocks:
  PRINT
  PRINT
  PRINT "Rescanning will wipe out any existing commands.
Continue? ";
  INPUT "",A$
  IF UCASE$(LEFT$(A$,1))="Y" THEN GOSUB BlockScan
  GOTO RangeManager

BlockScan:
  ON ERROR GOTO NoLayerFile
  OPEN PartLayerFileName$ FOR INPUT AS #3
  ON ERROR GOTO 0
  PRINT "Scanning ";PartLayerFileName$;"..."
  BlockIndex% = 0
  IF RangeCount%>0 THEN
```

```
    R%=1
    BlockStart%[R%] = BlockIndex%
    BlockEnd%[R%] = BlockIndex%-1

ScanReadBlocks:
    IF EOF(3) THEN ScanStartNewRange
    LINE INPUT #3,CONTROL$
ScanAlreadyHaveBlock:
    T$ = LEFT$(CONTROL$,1)
    IF LEN(CONTROL$)=0 OR T$="!" OR T$="*" THEN
ScanReadBlocks
'
'  parse out layer number, block name
'
    ControlIndex% = 1
    CALL
UnitValInt(ControlLayer&,CONTROL$,ControlIndex%)
    CALL UnitStr(ControlType$,CONTROL$,ControlIndex%)
    T$ = FN NoSp$(MID$(CONTROL$,ControlIndex%))
    I% = INSTR(T$,"!")
    IF I%=0 THEN I%=LEN(T$)+1
    ControlCmd$ = FN NoSp$(LEFT$(T$,I%-1))

    IF ControlType$="#CA" THEN ScanReadBlocks    ' skip
critical areas
    IF ControlLayer& < ZStart&[R%] THEN ScanReadBlocks
    IF ControlLayer& > ZEnd&[R%] THEN ScanStartNewRange
'
'  add unique blocks to current range's block table
'
    FOR J%=BlockStart%[R%] TO BlockEnd%[R%]
      IF ControlType$=BlockType$[J%] THEN ScanReadBlocks
    NEXT J%
    BlockType$[BlockIndex%] = ControlType$
    BlockCmd$[BlockIndex%] = ControlCmd$
    BlockEnd%[R%] = BlockIndex%
    INCR BlockIndex%
```

```
    GOTO ScanReadBlocks
'
'  set up the new layer
'
ScanStartNewRange:
    INCR R%
    IF R%<=RangeCount% THEN
      BlockStart%[R%] = BlockIndex%
      BlockEnd%[R%] = BlockIndex%-1
      GOTO ScanAlreadyHaveBlock
    END IF
  END IF
  CLOSE #3
  RETURN

'-----------------------------------------------------

MakeReport:
  PRINT
  PRINT
  PRINT "Press V to view report on screen, or P to print
on printer: ";
LoopReport:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopReport
  IF C$="V" THEN ViewReport
  IF C$="P" THEN PrintReport
  GOTO RangeManager

ViewReport:
  CLS
  T$ = " PREPARE Report for "+PartLayerFileName$
  T$ = T$ + STRING$(59-LEN(T$)," ") + DATE$+"  "+TIME$
  FOR I%=1% TO RangeCount%
    CLS
    PRINT T$
    PRINT
```

```
    PRINT " Range   Z-Start   Z-End   Z-Space
CAD-Start    CAD-End    CAD-Space   Count"
    PRINT USING " ### ";I%;
    PRINT USING "     ##### ";ZStart&[I%];
    PRINT USING "   #####";ZEnd&[I%];
    PRINT USING "    ##### ";ZSpace%[I%];
    PRINT USING "   #####.###";ZStart&[I%]/SliceRes;
    PRINT USING "   #####.###";ZEnd&[I%]/SliceRes;
    PRINT USING "   #####.###";ZSpace%[I%]/SliceRes;
    PRINT USING "
#####";(ZEnd&[I%]-ZStart&[I%])/ZSpace%[I%]+1
    PRINT "
----------------------------------------------------------"
    BlockCount% = 1
    FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
      PRINT "           Block";BlockCount%;
      PRINT "  Type
";BlockType$[J%];SPC(6-LEN(BlockType$[J%]));
      IF LEN(BlockCmd$[J%]) THEN PRINT "   Commands:
";BlockCmd$[J%];
      PRINT
      INCR BlockCount%
    NEXT J%
    PRINT
"----------------------------------------------------------"
    PRINT
    INPUT "Press Enter to continue...",A$
  NEXT I%
  GOTO RangeManager

PrintReport:
  ON ERROR GOTO PrinterNotReady
  LPRINT
  LPRINT
  T$ = " PREPARE Report for "+PartLayerFileName$
  LPRINT T$ + STRING$(59-LEN(T$)," ") + DATE$;"  ";TIME$
  LPRINT
```

```
  LPRINT " Range   Z-Start   Z-End   Z-Space   CAD-Start
  CAD-End    CAD-Space   Count"
  FOR I%=1% TO RangeCount%
    LPRINT USING " ### ";I%;
    LPRINT USING "     ##### ";ZStart&[I%];
    LPRINT USING "   #####";ZEnd&[I%];
    LPRINT USING "    ##### ";ZSpace%[I%];
    LPRINT USING "   #####.###";ZStart&[I%]/SliceRes;
    LPRINT USING "   #####.###";ZEnd&[I%]/SliceRes;
    LPRINT USING "   #####.###";ZSpace%[I%]/SliceRes;
    LPRINT USING "
#####";(ZEnd&[I%]-ZStart&[I%])/ZSpace%[I%]+1
    LPRINT "
----------------------------------------------------------"
    BlockCount% = 1
    FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
      LPRINT "           Block";BlockCount%;
      LPRINT "  Type
";BlockType$[J%];SPC(6-LEN(BlockType$[J%]));
      IF LEN(BlockCmd$[J%]) THEN LPRINT " Commands:
";BlockCmd$[J%];
      LPRINT
      INCR BlockCount%
    NEXT J%
    LPRINT
"----------------------------------------------------------"
  NEXT I%
  LPRINT CHR$(12);
  GOTO RangeManager

PrinterNotReady:
  PRINT
  PRINT "Printer not ready.  Press P to try again, or Q
to quit: ";
PrintNRLoop:
  DO
    C$ = UCASE$(INKEY$)
```

```
  LOOP UNTIL LEN(C$)>0
  IF C$="P" THEN RESUME
  IF C$="Q" THEN RESUME RangeManager
  GOTO PrintNRLoop

'-----------------------------------------------------
UpdateRangeFile:
  PRINT
  PRINT

  ON ERROR GOTO RangeFileProb
  OPEN PartRangeFileName$ FOR OUTPUT AS #10
  PRINT "Updating ";PartRangeFileName$;"..."
  FOR I%=1 TO CommentCount%
    PRINT #10,Comments$[I%]
  NEXT I%
  PRINT #10,"* ";
  WRITE #10,SliceRes,ZSpacing,Thickness
  FOR I%=1% TO RangeCount%
    WRITE
#10,ZStart&[I%],ZEnd&[I%],ZSpace%[I%],CureDepth%[I%]
    FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
      PRINT #10,BlockType$[J%];
      IF LEN(BlockCmd$[J%])>0 THEN PRINT
#10,",";Quote$;BlockCmd$[J%];Quote$;
      PRINT #10,""
    NEXT J%
  NEXT I%
  CLOSE #10
  GOTO RangeManager


'=====================================================
'
'  Layer Manager
'  - allows editing of individual layers
'           listing of layers having specified blocks
```

```
'
'
LayerManager:
  CLS
  IF RangeCount% = 0 THEN
    PRINT "No layer ranges!"
  ELSE
    T$ = "LAYER MANAGER"
    GOSUB CenterLine
    T$ = Title$
    GOSUB CenterLine
    T$ = "working on file "+PartLayerFileName$
    GOSUB CenterLine
    Indent$ = STRING$(18," ")
    PRINT
    PRINT Indent$;LayerBar$
    PRINT Indent$;" Range   Z-Start   Z-End   Z-Space
Count"
    PRINT Indent$;LayerBar$
    FOR I%=1 TO RangeCount%
      PRINT Indent$;
      PRINT USING " ### ";I%;
      PRINT USING "     ##### ";ZStart&[I%];
      PRINT USING "   #####";ZEnd&[I%];
      PRINT USING "    ##### ";ZSpace%[I%];
      IF ZSpace%[I%]>0 THEN
        Z% = (ZEnd&[I%]-ZStart&[I%])/ZSpace%[I%]
      ELSE
        Z% = 0
      END IF
      PRINT USING "   #####";Z%+1
    NEXT I%
    PRINT Indent$;LayerBar$
  END IF
  PRINT
  PRINT "   Press:"
```

```
  PRINT "    F  find blocks     E  edit layer file     U
update disk"
  PRINT "    X  exit            Q  quit program"
LoopLayer:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopLayer
  IF C$="F" THEN FindBlocks
  IF C$="E" THEN EditLayer
  IF C$="U" THEN UpdateLayerFile
  IF C$="X" THEN MainMenu
  IF C$="Q" THEN END
  GOTO LoopLayer

'-------------------------------------------------------

FindBlocks:
  PRINT
  PRINT
  INPUT "Find what blocks (separate by spaces)? ",B$
  IF LEN(B$)=0 THEN LayerManager
  TBIndex% = 0
  WHILE LEN(B$)>0
    I% = INSTR(B$," ")
    IF I%=0 THEN I%=LEN(B$)+1
    TBType$[TBIndex%] = UCASE$(LEFT$(B$,I%-1))
    INCR TBIndex%
    B$ = MID$(B$,I%+1)
  WEND
  PRINT "Look through what range of layers?  Enter ALL,
or enter number of the single"
  PRINT "range, or enter Z-Layer start,stop: ";
  LINE INPUT "",Z$
  IF LEN(Z$)=0 THEN LayerManager
  IF LEFT$(UCASE$(Z$),1)="A" THEN
    MinLayer& = 0
    MaxLayer& = 99999
  ELSE
```

```
    I% = INSTR(Z$,",")
    IF I% THEN
      MinLayer& = VAL(LEFT$(Z$,I%-1))
      MaxLayer& = VAL(MID$(Z$,I%+1))
    ELSE
      IF LEFT$(Z$,1)="#" THEN Z$=MID$(Z$,2)
      J% = VAL(Z$)
      MinLayer& = ZStart&[J%]
      MaxLayer& = ZEnd&[J%]
    END IF
  END IF

FindStart:
  ON ERROR GOTO NoLayerFile
  OPEN PartLayerFileName$ FOR INPUT AS #3
  ON ERROR GOTO 0
  PRINT "Searching ";PartLayerFileName$;"..."

FindLoop:
  IF EOF(3) THEN FindDone
  LINE INPUT #3,CONTROL$
  L$ = LEFT$(CONTROL$,1)
  IF LEN(CONTROL$)=0 OR L$="*" OR L$="!" THEN FindLoop
  ControlIndex% = 1
  CALL UnitValInt(ControlLayer&,CONTROL$,ControlIndex%)
  CALL UnitStr(ControlType$,CONTROL$,ControlIndex%)
  IF ControlLayer& < MinLayer& OR ControlLayer& >
MaxLayer& THEN FindLoop
  FOR I%=0 TO TBIndex%-1
    IF ControlType$=TBType$[I%] THEN
      PRINT TBType$[I%];" block found on
layer";ControlLayer&
      GOTO FindLoop
    END IF
  NEXT I%
  GOTO FindLoop
```

```
FindDone:
  CLOSE #3
  PRINT
  INPUT "Press Enter to continue... ",A$
  GOTO LayerManager

'-------------------------------------------------------

EditLayer:
  SHELL "EDIT "+PartLayerFileName$
  GOTO LayerManager

'-------------------------------------------------------

UpdateLayerFile:
  PRINT
  PRINT
  PRINT "About to update Layer File
";PartLayerFileName$;
  INPUT "  Okay? ",A$
  IF A$="" OR UCASE$(LEFT$(A$,1))<>"Y" THEN LayerManager
  ON ERROR GOTO NoBackup
  KILL BackupFileName$
  GOTO UpdateStart
NoBackup:
  RESUME UpdateStart

RenameError:
  PRINT "Error trying to rename ";PartLayerFileName$
  GotWarning% = TRUE
NoLayerControl:
  RESUME UpdateDone

UpdateStart:
  ON ERROR GOTO NoLayerControl
  NAME PartLayerFileName$ AS BackupFileName$
  ON ERROR GOTO NoBackupFile
```

```
  OPEN BackupFileName$ FOR INPUT AS #3
  ON ERROR GOTO NoLayerFile
  OPEN PartLayerFileName$ FOR OUTPUT AS #4
  ON ERROR GOTO 0
  PRINT "Updating ";PartLayerFileName$;"..."
  GotWarning% = FALSE

UpdateLoop:
  IF EOF(3) THEN UpdateDone
  LINE INPUT #3,CONTROL$
  L$ = LEFT$(CONTROL$,1)
  IF LEN(CONTROL$)=0 OR L$="*" OR L$="!" THEN
      PRINT #4,CONTROL$
      GOTO UpdateLoop
  END IF
  ControlIndex% = 1
  CALL UnitValInt(ControlLayer&,CONTROL$,ControlIndex%)
  CALL UnitStr(ControlType$,CONTROL$,ControlIndex%)
  T$ = FN NoSp$(MID$(CONTROL$,ControlIndex%))
  I%=INSTR(T$,"!")
  IF I%=0 THEN I%=LEN(T$)+1
  ControlCmd$ = FN NoSp$(LEFT$(T$,I%-1))
  ControlComment$ = MID$(T$,I%)

UpdateBlock:
  IF ControlType$="#CA" THEN
    PRINT #4,CONTROL$
    GOTO UpdateLoop
  END IF
  FOR I%=1 TO RangeCount%
    IF ControlLayer& < ZStart&[I%] THEN UpdLayer
  NEXT I%
  IF ControlLayer& <= ZEnd&[I%-1] THEN UpdLayer
  PRINT "Warning:  Layer";ControlLayer&;"not accounted
for in table."
BadUpdate:
  GotWarning% = TRUE
```

```
    PRINT #4,CONTROL$
    GOTO UpdateLoop

UpdLayer:
    DECR I%
    FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
      IF BlockType$[J%]=ControlType$ THEN UpdBlock
    NEXT J%
    PRINT "Warning:  Layer";ControlLayer&;"Block ";
     ControlType$;_
      " not accounted for in table."
    GOTO BadUpdate

UpdBlock:
    PRINT #4,FN
NoSp$(STR$(ControlLayer&));",";ControlType$;
    IF LEN(BlockCmd$[J%])>0 THEN PRINT
#4,",";BlockCmd$[J%];
    IF LEN(ControlComment$) THEN PRINT #4,"
       ";ControlComment$;
    PRINT #4,""
    GOTO UpdateLoop

UpdateDone:
    ON ERROR GOTO 0
    CLOSE #3
    CLOSE #4
    IF GotWarning% THEN
      INPUT "Press Enter to continue... ",A$
    END IF
    GOTO LayerManager


'=======================================================
'
'  Calculate part's step period(s)
'
```

```
CalcSp:
  ON ERROR GOTO NoPowerFile
  OPEN PowerFile$ FOR INPUT AS #8
  INPUT #8,LaserPower
  CLOSE #8
  GOTO CalcReadMaterialData
NoPowerFile:
  LaserPower = 0
  RESUME CalcReadMaterialData

CalcReadMaterialData:
  ON ERROR GOTO NoLastMaterialFile
  OPEN LastMaterialFile$ FOR INPUT AS #9
  LINE INPUT #9,MaterialFile$
  CLOSE #9
  GOSUB GetMaterialFileData
  GOTO ShowSp
NoLastMaterialFile:
  MaterialFile$ = "- none -"
  FOR I%=1 TO RangeCount%
    EstSp&[I%] = 0
    SP(I%)     = 0   :   LH(I%)    = 0   :   WMIN(I%) = 0
    WMAX(I%) =  0
    LHSlope    = 0   :   LHYInt    = 0   :   LHRval   = 0
    WMINSlope = 0    :   WMINYint = 0    :   WMINRval = 0
    WMAXSlope = 0    :   WMAXYint = 0    :   WMAXRval = 0
  NEXT I%
  RESUME ShowSp

ShowSp:
  CLS
  ON ERROR GOTO 0
  IF RangeCount% = 0 THEN
    PRINT "No layer ranges!"
  ELSE
    T$ = "RANGE - CALC SP"
    GOSUB CenterLine
```

```
    T$ = Title$
    GOSUB CenterLine
    T$ = "working on file "+PartRangeFileName$
    GOSUB CenterLine
    Indent$ = STRING$(6," ")
    PRINT
    T$ = "Laser Power Reading:
"+STR$(INT(LaserPower*100)/100)+" mW"
    GOSUB CenterLine
    T$ = "using working curve data from material file:
"+MaterialFile$
    GOSUB CenterLine
    PRINT Indent$;SpBar$
    PRINT Indent$;" Range   Z-Start   Z-End   Z-Space
Depth   Est SP   W Min   W Max"
    PRINT Indent$;SpBar$
    FOR I%=1 TO RangeCount%
      WMIN = FN GetLog(EstSp&[I%],WMINSlope,WMINYint)
      IF WMIN>999 THEN WMIN=999
      IF WMIN<0 THEN WMIN=0
      WMAX = FN GetLog(EstSp&[I%],WMAXSlope,WMAXYint)
      IF WMAX>999 THEN WMAX=999
      IF WMAX<0 THEN WMAX=0
      PRINT Indent$;
      PRINT USING " ### ";I%;
      PRINT USING "     ##### ";ZStart&[I%];
      PRINT USING "   #####";ZEnd&[I%];
      PRINT USING "   ##### ";ZSpace%[I%];
      PRINT USING "   #####";CureDepth%[I%];
      PRINT USING "   #####";EstSp&[I%];
      PRINT USING "     ###.#";WMIN;
      PRINT USING "   ###.#";WMAX
    NEXT I%
    PRINT Indent$;SpBar$
  END IF
  PRINT
  PRINT "   Press:"
```

```
  PRINT "    R  read material data      S  save new
                                        range commands"
  PRINT "    P  change power rating     E  edit cure
                                       depth/step period"
  PRINT "    V  view material data"
  PRINT "    X  exit                    Q  quit program"
LoopSp:
  C$=UCASE$(INKEY$)
  IF LEN(C$)=0 THEN LoopSp
  IF C$="R" THEN ReadMaterial
  IF C$="P" THEN ChangeSp
  IF C$="S" THEN UpdateCmds
  IF C$="V" THEN ViewMaterial
  IF C$="E" THEN EditSpParams
  IF C$="X" THEN RangeManager
  IF C$="Q" THEN END
  GOTO LoopSp

'-----------------------------------------------------

ReadMaterial:
  CLS
  PRINT "Available Material Files:"
  PRINT
  ON ERROR GOTO NoMaterialFiles
  FILES "*.MAT"
  GOTO GetMaterialAsk
NoMaterialFiles:
  PRINT "- no material files in working directory -"
  RESUME GetMaterialAsk
GetMaterialAsk:
  PRINT
  INPUT "Name of data file to read: ",MaterialFile$
  IF LEN(MaterialFile$)=0 THEN ShowSp
  MaterialFile$ = UCASE$(MaterialFile$)
  IF INSTR(MaterialFile$,".")=0 THEN MaterialFile$ =
MaterialFile$ + ".MAT"
```

```
    ON ERROR GOTO .CantReadMaterialFile
    GOSUB GetMaterialFileData
    OPEN LastMaterialFile$ FOR OUTPUT AS #9
    PRINT #9,MaterialFile$
    CLOSE #9
    GOTO ShowSp

GetMaterialFileData:
    OPEN MaterialFile$ FOR INPUT AS #7
    ON ERROR GOTO 0
    INPUT #7,MaterialLaserPower
    INPUT #7,NumberOfPairs%
    FOR I% = 1 TO NumberOfPairs%
      INPUT #7,SP(I%)
      INPUT #7,LH(I%)
      SP(I%) = LOG(SP(I%))/LOG(10)
      INPUT #7,WMIN(I%)
      INPUT #7,WMAX(I%)
    NEXT I%
    CLOSE #7
    CALL Regression(LH(),LHSlope,LHYint,LHRval)
    CALL Regression(WMIN(),WMINSlope,WMINYint,WMINRval)
    CALL Regression(WMAX(),WMAXSlope,WMAXYint,WMAXRval)
    GOSUB DoCalculation
    RETURN

CantReadMaterialFile:
    PRINT "Can't read material file ";MaterialFile$
    CLOSE #7
    RESUME GetMaterialAsk

'-----------------------------------------------------

ChangeSp:
    PRINT
    INPUT "Enter new laser power reading (mW):
",LaserPower
```

```
  OPEN PowerFile$ FOR OUTPUT AS #8
  PRINT #8,LaserPower
  CLOSE #8
  GOSUB DoCalculation
  GOTO ShowSp

'-------------------------------------------------------

UpdateCmds:
  CLS
  PRINT "Update Range Commands"
  PRINT
  PRINT "Block Groups:"
  PRINT
  FOR I%=0 TO BlockGroups%
    PRINT FN StrNoSp$(I%);".  ";
    FOR J%=0 TO BlockGroupCount%[I%]-1
      IF POS(0)>75 THEN
        PRINT
        PRINT "    ";
      END IF
      PRINT
BlockGroupId$[I%,J%];SPC(6-LEN(BlockGroupId$[I%,J%]));
    NEXT J%
    PRINT
  NEXT I%
  PRINT
  INPUT "Use which block group [1]? ",A$
  IF LEN(A$)=0 THEN Group% = 1 ELSE Group% = VAL(A$)
  PRINT
  INPUT "Update which merge-sets [all]? ",Upd$
  Upd$ = UCASE$(Upd$)
  IF LEN(Upd$)=0 OR LEFT$(Upd$,1)="A" THEN Upd$="A"
  FOR I%=1 TO RangeCount%
    NewSp$ = "SP " + FN NoSp$(STR$(EstSp&[I%]))
    FOR J%=BlockStart%[I%] TO BlockEnd%[I%]
      B$ = BlockType$[J%]
```

```
        F$ = ""
        WHILE RIGHT$(B$,1)<"A"
          F$ = F$+RIGHT$(B$,1)
          B$ = LEFT$(B$,LEN(B$)-1)
        WEND
        IF B$="L" OR (Upd$<>"A" AND INSTR(Upd$,F$)=0) THEN
NoUpdateCmdBlock
        K%=0
        WHILE K% < BlockGroupCount%[Group%]
          IF B$=BlockGroupId$[Group%,K%] THEN
UpdateCmdBlock
          INCR K%
        WEND
        GOTO NoUpdateCmdBlock
UpdateCmdBlock:
        IF RIGHT$(B$,1)="F" THEN
          NewSs$ = FillStepSize$
        ELSE
          NewSs$ = NormStepSize$
        END IF
        IF NewSs$<>"" THEN CALL
UpdateBlockCommands(J%,B$,NewSs$)
        IF NewSp$<>"" THEN CALL
UpdateBlockCommands(J%,B$,NewSp$)

NoUpdateCmdBlock:
    NEXT J%
  NEXT I%
  PRINT
  PRINT "Range commands updated in memory.  Use Range
Manager Update Disk Command"
  INPUT "to update the Range File out on disk...",A$
  GOTO ShowSp


SUB UpdateBlockCommands(Index%,Block$,NewItem$)
LOCAL T$,ItemId$,I%,J%,K%,CmdLeft$,CmdRight$
```

```
SHARED BlockCmd$[]
  I% = INSTR(NewItem$," ")    ' get text up to first
space (or end)
  IF I%=0 THEN I%=LEN(NewItem$)+1
  ItemId$ = LEFT$(NewItem$,I%-1)
  T$ = FN NoQuotes$(BlockCmd$[Index%])
  IF LEN(T$)=0 THEN
    T$ = NewItem$
  ELSE
    J% = INSTR(T$,ItemId$)
    IF J% THEN
      K% = INSTR(J%,T$,";")
      IF K%=0 THEN K% = LEN(T$)
      CmdLeft$ = LEFT$(T$,J%-1)
      CmdRight$ = MID$(T$,K%+1,99)
      IF LEN(CmdRight$)=0 AND RIGHT$(CmdLeft$,1)=";"
THEN_
        CmdLeft$ = LEFT$(CmdLeft$,LEN(CmdLeft$)-1)
      T$ = CmdLeft$+CmdRight$
    END IF
    IF LEN(T$) THEN
      T$ = NewItem$ + ";" + T$
    ELSE
      T$ = NewItem$
    END IF
  END IF
  IF LEN(T$) THEN T$ = CHR$(34) + T$ + CHR$(34)
  BlockCmd$[Index%] = T$
UpdateBlocksEnd:
END SUB

'-----------------------------------------------------

ViewMaterial:
  CLS
  PRINT "Data from Material File: ";MaterialFile$
  PRINT
```

```
PRINT
PRINT "Material Test Laser Power Reading: ";
PRINT USING "###.##";MaterialLaserPower
PRINT
PRINT NumberOfPairs%;"Material Data Pairs:"
PRINT
FOR I% = 1 TO NumberOfPairs%
  PRINT USING "##. "; I%;
  PRINT USING "  SP = ###.##";10^SP(I%);
  LOCATE ,20
  PRINT USING "  LH = ####.##";LH(I%);
  LOCATE ,35
  PRINT USING "  WMIN = ####.##";WMIN(I%);
  LOCATE ,52
  PRINT USING "  WMAX = ####.##";WMAX(I%)
NEXT I%
PRINT
PRINT "  LH:  Slope =";
PRINT USING "#####.##";INT(LHSlope*1000)/1000;
PRINT "   Y-Intercept =";
PRINT USING "#####.##";INT(LHYint*100)/100;
PRINT "   R-Value =";
PRINT USING "#####.##";INT(LHRval*1000)/1000
PRINT "WMIN:  Slope =";
PRINT USING "#####.##";INT(WMINSlope*1000)/1000;
PRINT "   Y-Intercept =";
PRINT USING "#####.##";INT(WMINYint*100)/100;
PRINT "   R-Value =";
PRINT USING "#####.##";INT(WMINRval*1000)/1000
PRINT "WMAX:  Slope =";
PRINT USING "#####.##";INT(WMAXSlope*1000)/1000;
PRINT "   Y-Intercept =";
PRINT USING "#####.##";INT(WMAXYint*100)/100;
PRINT "   R-Value =";
PRINT USING "#####.##";INT(WMAXRval*1000)/1000
PRINT
INPUT "Press Enter to continue... ",A$
```

```
  GOTO ShowSp

'-----------------------------------------------------

EditSpParams:
  PRINT
  PRINT "Enter C followed by range #,new cure depth
  PRINT "   or S followed by range #,new step period: ";
  LINE INPUT "",A$
  C$ = UCASE$(LEFT$(A$,1))
  IF C$="S" THEN
    ModingSp% = TRUE
    GOTO EditCont
  ELSEIF C$="C" THEN
    ModingSp% = FALSE
    GOTO EditCont
  END IF
  GOTO ShowSp
EditCont:
  A$ = MID$(A$,2,99)
  IF INSTR(A$,",")=0 THEN ShowSp
  I% = INSTR(A$,",")
  Range% = VAL(LEFT$(A$,I%-1))
  IF Range%<1 OR Range%>RangeCount% THEN ShowSp
  Value# = VAL(MID$(A$,I%+1))
  IF Value# <= MaxInt THEN
    Value% = Value#
    IF ModingSp% THEN
      EstSp&[Range%] = Value%
      CureDepth%[Range%] = FN
GetLog(Value%,LHSlope,LHYint)
    ELSE
      CureDepth%[Range%] = Value%
      EstSp&[Range%] = FN GetExp(Value%,LHSlope,LHYint)
    END IF
    IF EstSp&[Range%] < MinSp& THEN EstSp&[Range%] =
MinSp&
```

```
    IF EstSP&[Range%] > MaxSp& THEN EstSp&[Range%] =
MaxSp&
  END IF
  GOTO ShowSp

DoCalculation:
  FOR I%=1 TO RangeCount%
    IF LaserPower < 1 THEN EstSp&[I%] = MinSp&_
    ELSE EstSp&[I%] = FN
GetExp(CureDepth%[I%],LHSlope,LHYint)
    IF EstSp&[I%] < MinSp& THEN EstSp&[I%] = MinSp&
    IF EstSP&[I%] > MaxSp& THEN EstSp&[I%] = MaxSp&
  NEXT I%
  RETURN

'-------------------------------------------------------

SUB Regression(Array(1),Slope,Yint,Rval)
  SHARED NumberOfPairs%,SP()
  XY        = 0
  SUMX      = 0
  SUMY      = 0
  SUMY2     = 0
  XSQUARED = 0
  IF NumberOfPairs%<1 THEN
    Slope = 0
    Yint  = 0
    Rval  = 0
    GOTO EndRegSub
  END IF
  FOR I% = 1 TO NumberOfPairs%
    XY = XY + SP(I%) * Array(I%)
    SUMX = SUMX + SP(I%)
    SUMY = SUMY + Array(I%)
    SUMY2 = SUMY2 + Array(I%)^2
    XSQUARED = XSQUARED + SP(I%)^2
  NEXT I%
```

```
  Denom = XSQUARED - (SUMX^2) / NumberOfPairs%
  IF Denom < 1E-5 THEN Slope = 0_
  ELSE Slope = (XY - (SUMX*SUMY)/NumberOfPairs%) / Denom
  Yint = SUMY/NumberOfPairs% - Slope *
SUMX/NumberOfPairs%
  Denom = SUMY2 - SUMY^2 / NumberOfPairs%
  IF Denom < 1E-5 THEN Rval = 0_
  ELSE Rval = SQR(((Yint * SUMY + Slope * XY) -
SUMY^2/NumberOfPairs%) / Denom)
EndRegSub:
END SUB

DEF FN GetExp(DataPoint,Slope,Yint)
  IF (LaserPower>0) AND (Slope>0) THEN
    temp# = 10 ^ ((DataPoint - Yint) / Slope) *
MaterialLaserPower / LaserPower
    IF temp# <= MaxUInt THEN FN GetExp = temp# ELSE FN
GetExp = MaxUInt
  ELSE
    FN GetExp = 0
  END IF
END DEF

DEF FN GetLog(DataPoint,Slope,Yint)
  IF (DataPoint>0) AND (MaterialLaserPower>0) AND
(LaserPower>0) THEN
    FN GetLog = Slope *
LOG10(DataPoint/(MaterialLaserPower / LaserPower)) +
Yint
  ELSE
    FN GetLog = 0
  END IF
END DEF


'=========================================================
```

```
'  Statistics:
'    show full directory of all files involved with part
'   Statistics:
  CLS
  PRINT "Statistics for part ";PartName$;":"
  SHELL "DIR "+PartName$+".*"
  LOCATE 23,1
  INPUT "Press Enter to continue: ",A$
  GOTO MainMenu


'=============================================================
'
'  UTILITY FUNCTIONS AND PROCEDURES
'
'-------------------------------------------------------------
'
' Removes all double spaces in CmdLine$
'
RemoveDoubleSpaces:
  WHILE INSTR(CmdLine$,"  ")>0
    I = INSTR(CmdLine$,"  ")
    CmdLine$ = left$(CmdLine$,I) + mid$(CmdLine$,I+2,99)
  WEND
  RETURN

'
'  Find any char of Char$ in Main$ starting at index
Start
'  Returns LEN(Main$)+1 if no character of Char$ in
Main$
'
DEF FN FindStr(Start%,Main$,Char$)
  LOCAL I%,J%,MIN%
  MIN% = LEN(Main$) + 1
  FOR I%=1 to LEN(Char$)
    J% = INSTR(Start%,Main$,MID$(Char$,I%,1))
```

```
    IF (J% > 0) AND (J% < MIN%) THEN MIN% = J%
  NEXT I%
  FN FindStr = MIN%
END DEF

'
'  Make a string from a number without a leading space
'
DEF FN StrNoSp$(Num)
  LOCAL A$
  A$=STR$(Num)
  IF LEFT$(A$,1)=" " THEN A$=RIGHT$(A$,LEN(A$)-1)
  FN StrNoSp$ = A$
END DEF

'
'  Pick out a number from a string
'  starting at Index and going to the next space or
comma or end-of-string
'
SUB UnitValShortInt(Storage%,Text$,Index%)
  Storage% = VAL(MID$(Text$,Index%))
  Index% = FN FindStr(Index%,Text$," ,") + 1
END SUB

SUB UnitValInt(Storage&,Text$,Index%)
  Storage& = VAL(MID$(Text$,Index%))
  Index% = FN FindStr(Index%,Text$," ,") + 1
END SUB

SUB UnitVal(Storage,Text$,Index%)
  Storage = VAL(MID$(Text$,Index%))
  Index% = FN FindStr(Index%,Text$," ,") + 1
END SUB

'
'  Pick out text from a string
```

```
'  starting at Index and going to the next comma or
end-of-string
'
SUB UnitStr(Storage$,Text$,Index%)
  LOCAL I%
  I% = FN FindStr(Index%,Text$," ,")
  Storage$ = MID$(Text$,Index%,I%-Index%)
  Index% = I% + 1
END SUB

'
'  Remove excess quote marks around string
'
DEF FN NoQuotes$(Text$)
  LOCAL S%,E%
  S% = 1
  IF LEFT$(Text$,1)=CHR$(34) THEN INCR S%
  E% = LEN(Text$)+1
  IF RIGHT$(Text$,1)=CHR$(34) THEN DECR E%
  FN NoQuotes$ = MID$(Text$,S%,E%-S%)
END DEF

'
'  Remove excess spaces around string
'
DEF FN NoSp$(Text$)
  LOCAL S%,E%
  IF LEN(Text$)>0 THEN
    S% = 1
    WHILE MID$(Text$,S%,1)=" "
      INCR S%
    WEND
    E% = LEN(Text$)
    WHILE E%>S%+1 AND MID$(Text$,E%,1)=" "
      DECR E%
    WEND
    FN NoSp$ = MID$(Text$,S%,E%-S%+1)
```

```
  ELSE
    FN NoSp$ = ""
  END IF
END DEF

'
'  Can't open file.  Report this to operator.
'
CantOpen:
  CALL SystemError("Can't open file "+FileName$)
  END

'-------------------------------------------------------
'
'  Show error and store it on disk
'
SUB SystemError(Text$)
  PRINT Text$
  OPEN "C:\SYSTEM.ERR" FOR OUTPUT AS #9
  PRINT #9,"PREPARE: ";Text$
  CLOSE #9
END SUB

CenterLine:
  LOCATE ,(79-LEN(T$))/2
  PRINT T$
  RETURN
^Z
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================}
   SERVICE.PAS
   SLA-1 Special Services Provider
---------------------------------------------------------
   Recent History:
   11/20/87  Ver 2.30  under development
   12/06/87  Ver 2.31  Error output shows SERVICE, not
                       MERGE
    /I in command line causes all errors to be ignored
    /T in command line for transparent operation
    1/25/88  Ver 2.60  version update only
    2/24/88  Ver 2.61  converted to Turbo Pascal Version
                       4.0 added program title header
                       added invalid command error
    3/09/88  Ver 2.80  added copyright notice
    4/09/88  Ver 3.00  production release

---------------------------------------------------------}
uses Dos;
const
  VersionId    = '3.00';
  MSDOS        = $21;   { interrupt vectors }
  LASERVECTOR = $64;
  ErrorFileName = 'C:\SYSTEM.ERR';
type
  str4 = string[4];
  str16 = string[16];

var
  i: Integer;
  ErrorFile: Text;
  regs: Registers;
  LaserVersion: str4;
  SingleBar,CmdLine,cmd: String;
  ShowErrors,NoText: Boolean;
```

```
function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function CleanUpSpaces(Text: String): String;
var
  Temp: String;
  i: Integer;
  HaveSpace: Boolean;
begin
  i := 1;
  while (i < length(Text)) and (copy(Text,1,1)=' ') do
   inc(i);
  HaveSpace := false;
  Temp := '';
  while i <= length(Text) do begin
    if Text[i]=' ' then HaveSpace := true
    else begin
      if HaveSpace then begin
        Temp := Temp + ' ' + Text[i];
```

```
        HaveSpace := false;
      end else Temp := Temp + Text[i];
    end;
    inc(i);
  end; {while i}
  CleanUpSpaces := Temp;
end;

{
   convert integer to a string
}
function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

{
   Tell user if not transparent
}
procedure Tell(Text: String);
begin
  if not NoText then writeln(Text);
end;

{
   read DOS command line
}
procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
```

```
end;

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln(Text);
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'SERVICE: ',Text);
  close(ErrorFile);
end;

{-------------------------------------------------
                          Laser Controller Routines
-------------------------------------------------}
{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
```

```
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
    if LaserOk then LaserPresent := true
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{-------------------------------------------------------}
procedure TurnLaserControllerOff;
var
  Er: Integer;
begin
  with regs do begin
    AX := $0100;
    Intr(LASERVECTOR,regs);
    Er := AL;
    if ShowErrors then begin
      if Er=0 then Tell('Laser Controller turned off.')
      else Tell('Error'+IntStr(Er)+' attempting to turn
laser off.');
    end;
  end;
end;
procedure TurnLaserControllerOn;
var
  Er: Integer;
```

```
begin
  with regs do begin
    AX := $0000;
    BX := 1190;   { 1 ms step period }
    Intr(LASERVECTOR,regs);
    Er := AL;
    if ShowErrors then begin
      if Er=0 then Tell('Laser Controller turned on.')
      else Tell('Error'+IntStr(Er)+' attempting to turn
laser on.');
    end;
  end;
end;

procedure ShowHeader;
begin
  SingleBar := many(chr(196),45);
  writeln;
  writeln(SingleBar);
  writeln('  SERVICE, SLA Resource Utility');
  writeln('  Version ',VersionId);
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography
System');
  writeln(SingleBar);
  writeln;
end;

{ - - - M A I N   P R O G R A M - - - }

begin
  ShowErrors := true;
  NoText     := false;
  ReadCommandLine(CmdLine);
  cmd := CleanUpSpaces(upper(CmdLine));
  i := pos('/I',cmd);
  if i>0 then begin
```

```
    ShowErrors := false;
    delete(cmd,i,2);
  end;
  i := pos('/T',cmd);
  if i>0 then begin
    NoText := true;
    delete(cmd,i,2);
  end;
  if not NoText then ShowHeader;
  if not LaserPresent then halt(1);
  if length(cmd)>0 then begin
    if copy(cmd,1,5)='LASER' then begin
      if copy(cmd,7,2)='ON' then TurnLaserControllerOn
      else if copy(cmd,7,3)='OFF' then
TurnLaserControllerOff
      else Tell('Invalid LASER request.');
    end else Tell('Unrecognized command.');
  end;
  halt(0);
end.
```

SLICE.C

```
/*******************************************************
*   3D System's Unix Slice User Interface Program
*
*=======================================================
*
*   Version 2.40  Second Beta Release
*   Version 3.00  first production Release
*
*******************************************************/

#include <stdio.h>
#include :string.h>
#include <math.h>

#include <sys/types.h>
#include <sys/times.h>
struct tms timestamp1,timestamp2;
unsigned long time_val1,time_val2;
unsigned long system_time;

#include "slice.h"

#define   UI_FILE_EXT       ".UII"
#define   Z_SPACE_EXT       ".UIZ"
#define   SLICE_INPUT_EXT   ".stl"
#define   SLICE_OUTPUT_EXT  ".sli"
#define   LAST_FILE         "LAST.UIN"

#define   PARA_NO     11
#define   FUNC_NO     24
#define   FLUSH_IN    fflush(stdin)
#define   CLRSCR      system("tput clear")

#define   SLICE_PGM   "slice"
#define   SLICE_VER   "300"
```

```
#define   VERSION     3.00
#define   log_file_name "LOG.UI"
FILE *long_file_ptr;

/*******************************************************/

typedef struct
{
  int z_layer;
  int z_value;
} z_record;
/*******************************************************/
unsigned scaled(user_infor,num)
  USER_INFO_STRUC *user_info;
  float           num;
  {
  return((unsigned) (user_info->resolution*num+0.5));
  };
/*******************************************************/
char exact(user_info,num)
  USER_INFO_STRUC *user_info;
  float           num;
  {
  if (fabs(user_info->resolution * num - scaled(user_
  info,num)) < 1E-3)
    return(' ');
  else return('*');
  };

/*******************************************************/

wait_key()
{
  char line[2];
  printf("\n--- Press <Enter> to continue ---");
  getline(line,1);
}
```

```
/*******************************************************/
unsigned read_int()
{
  char line[7];
  int i;
  float temp;

  do {
    getline(line,6);
    i=0;
    parse_number(line,&i,&temp);
    if (temp < 0.0) temp =-temp;
    if (temp > 65535.0) printf("Number Range Error
     (0-65535). Retype: ");
  } while ( temp > 65535.0);
  return(temp);
}
/*******************************************************/
float read_float(check)
  BOOLEAN check;
{
  char line[13];
  int i;
  float temp;

  do {
    getline(line,12);
    i=0;
    parse_number(line,&i,&temp);
    if (temp < 0.0) temp = -temp;
    temp = (unsigned long) (temp*1000+0.0005) / (float)
     (1000);
    if (check && (temp > 65535.0)) printf("Number Range
     Error (0-65535).  Retype:
  } while (check && (temp > 65535.0));
  return(temp);
}
```

```
/*******************************************************/

openfile (filename,file_ptr,open_mode)
char  filename[];
FILE  **file_ptr;
char  *open_mode;

{
if  ((*file_ptr = fopen (filename, open_mode)) == NULL)
  {
  fprintf (stderr, "\n ERROR - unable to open %s\n,
   filename);
  exit (1);
  }
}  /* openfile */

/*******************************************************/

get_object_name (argc,argv,object_name)
int  argc;
char   *argv[];
char   object_name[];

{
  FILE  *file_ptr;

  if (argc > 1)
  {
    strcpy (object_name,k argv[1]);
    if (strchr(object_name,'.') := NULL)
    {
      fprintf (stderr, "\n*** Object name cannot
       contain periods.  ");
      fprintf (stderr, "Object name is not the file
       name ***\n");
      exit (1);
  }
```

```
} else
{
  if (!file_exist(LAST_FILE))
  {
    fprintf (stderr, "\n*** Cannot find %s.  Please use
     ",LAST_FILE);
    fprintf (stderr, "%s command with object name ***\n"
     ,argv[0]);
    exit (1);
  }
} else
{
  if (! file_exist (LAST_FILE))
{
  fprintf(stderr, "\n**Cannot find %s.
    Please use ",LAST_FILE);
  fprintf(stderr, "%s command with object name
    ***\n", argv[o]);
  exit(1);
}
    openfile (LAST_FILE, &file_ptr, "r");
    fscanf   (file_ptr, "%s", object_name);
    fclose   (file_ptr);
  }
} /* get_object_name */

/*******************************************************/

file_exist (file_name)
char      file_name[];
{
  FILE  *file_ptr;

  if ((file_ptr = fopen (file_name, "r" )) != NULL)
  {
    fclose(file_ptr);
    return(TRUE);
```

```
    }
    else return(FALSE);
} /* file_exist */

/*****************************************************/

update_last_UI (object_name)
  char    object_name[];
{
  FILE    *file_ptr;

  openfile (LAST_FILE, &file_ptr, "w");
  fprintf  (file_ptr, "%s\n", object_name);
  fclose(file-ptr);
}; /* update_last_UI */
/*****************************************************/
void read_UI_file(filename, user_info)
  char filename[];
  USER_INFO_STRUC *user_info;
{
  FILE      *file_ptr;
  char      option[STR_LEN],temp_str[STR_LEN];
  unsigned int i;

  openfile (filename, &file_ptr, "r");

while (fscanf(file_ptr,"%s",option) == 1) {
  if (strcmp(option,VERSION_TXT) == 0)
    fscanf(file_ptr,"%f",&user_info->version);
  if (strcmp(option,IN_FILE_TXT) == 0)
    fscanf(file_ptr,"%s",user_info->input_filename);
  if (strcmp(option,EXTRA_PARAMS_TXT) == 0) {
    fgets(user_info->extra_parameters,STR_LEN,file_ptr);
    user_info->extra_parameters[strlen(user_info->extra
     _parameters)-1] = '\0';
  }
  if (strcmp(option,OUT_FILE_TXT) == 0)
```

```
        fscanf(file_ptr,"%s",user_info->segfile_name);
      if (strcmp(option,SCALE_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->resolution);
      if (strcmp(option,RES_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->resolution);
      if (strcmp(option,Z_SPACE_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->z-spacing);
      if (strcmp(option,X_FLAT_FILL_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->x_flat_fill_
         spacing);
      if (strcmp(option,Y_FLAT_FILL_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->y_flat_fill_
         spacing);
      if (strcmp(option,X_HATCH_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->x_hatch_spacing);
      if (strcmp(option,Y_HATCH_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->y_hatch_spacing);
      if (strcmp(option,ANGLE_HATCH_TXT) == 0)
        fscanf(file_ptr,"%f",&user_info->angle_hatch_
    spacing);
      if (strcmp(option,MSA_TXT) == 0)
        fscanf(file_ptr,"%u",&user_info->min_surface_angle);
      if (strcmp(option,MIA_TXT) == 0)
        fscanf(file_ptr,"%u",&user_info->min_intersect_
         angle);
      if (strcmp(option,FILETYPE_TXT) == 0)
        fscanf(file_ptr,"%u",&user_info->file_type);
      if (strcmp(option,CENTERING_TXT) == 0)
        fscanf(file_ptr,"%u",&user_info->centering);
      if (strcmp(option,VAR_Z_SPACE_TXT) == 0) {
        i = 0;
        fscanf(file_ptr,"%s",temp_str);
        while (temp_str[0] != '*') {
          fscanf(file_ptr,"%f %f %s",&user_info->z_level[i],
                                &user_info->z_level_spacing[i],
                                temp_str);
            i++;
```

```
        }
      }
    }
    fclose(file_ptr);
    if (user_info->version < 2.339) {
      user_info->z_spacing /= user_info->resolution;
      user_info->x_flat_fill_spacing /= user_info->
       resolution;
      user_info->y_flat_fill_spacing /= user_info->
       resolution;
      user_info->x_hatch_spacing /= user_info->resolution;
      user_info->y_hatch_spacing /= user_info->resolution;
      user_into->angle_hatch_spacing /= user_info->
       resolution;
      };
    };

/*******************************************************/

get_user_info_from_file (user_info.object_name)
  USER_INFO_STRUC   *user_info;
  char              object_name[];
{
  char  filename[STR_LEN];

  strcpy(filename,object_name);
  strcat(filename,UI_FILE_EXT);
  init_user_info(user_info(user_info,object_name);
  if (file_exist(filename)) read_UI_file(filename,user_
   info);
} /* get_user_info_from_file */

/*******************************************************/

void save_UI_data_file(user_info,object_name)
  USER_INFO_STRUC   *user_info;
  char              object_name[];
```

```
{
  FILE          *file_ptr;
  char          filename[STR_LEN];
  unsigned int  i;

  strcpy(filename,object_name);
  strcat(filename,UI_FILE_EXT);
  openfile (filename, &file_ptr, "w");

  fprintf(file_ptr,"%s %f\n",VERSION_TXT,VERSION);
  fprintf(file_ptr,"%s %s\n",IN_FILE_TXT,user_info->
   input_filename);
  fprintf(file_ptr,"%s %s\n",EXTRA_PARAMS_TXT,user_
   info->extra_parameters);
  fprintf(file_ptr,"%s %s\n",OUT_FILE_TXT,user_
   info->segfile_name);
  fprintf(file_ptr,"%s %f\n",RES_TXT_,user_info->
   resolution);
  fprintf(file_ptr,"%s %f\n",Z_SPACE_TXT,user_info->
   z_spacing);
  fprintf(file_ptr,"%s %f\n",X_FLAT_FILL_TXT,user_
   info->x_flat_fill_spacing);
  fprintf(file_ptr,"%s %f\n",Y_FLAT_FILL_TXT,user_
   info->y_flat_fill_spacing);
  fprintf(file_ptr,"%s %f\n",X_HATCH_TXT,user_info->
   x_hatch_spacing);
  fprintf(file_ptr,"%s %f\n",Y_HATCH_TXT,user_info->
   y_hatch_spacing);
  fprintf(file_ptr,"%s %f\n",ANGLE_HATCH_TXT,user_
   info->angle_hatch_spacing);
  fprintf(file_ptr,"%s %u\n",MSA_TXT,user_info->min_
   surface_angle);
  fprintf(file_ptr,"%s %u\n",MIA_TXT,user_info->min_
   intersect_angle);
  fprintf(file_ptr,"%s %u\n",FILETYPE_TXT,user_info->
   _file_type);
```

```
  fprintf(file_ptr,"%s %u\n",CENTERING_TXT,user_info->
   centering);
  fprintf(file_ptr,"s\n",VAR_Z_SPACE_TXT);
  i = 0;
  while (user_info->z_level[i] >= 0) {
    fprintf(file_ptr," > %f %f\n",user_info->z_level[i],
    user_info->z_level_spacing;
    i++;
  }
  fprintf(file_ptr,: *\n");
  fclose(file_ptr);
  printf("User Interface Options for %s Saved\n",object_
   name);
  wait_key();
}  /* save_UI_data_file */

/*******************************************************/

show_main_menu(user_info,object_name)
  USER_INFO_STRUC *user_info;
  char            object_name[];

  char  filename [STR_LEN];
  unsigned int i;

  CLRSCR;
  printf ("               - SLICE USER INTERFACE -\n");
  printf ("              Version %-1.2f\n\n\n",VERSION);
  printf ("    Current Object: %s\n",object_name);
  printf ("Database File Name: %s",user_info->input
    _filename);
  switch (user_info->file_type) {
    case 0:
      printf (", ASCII format");
      break;
    case 1:
      printf (", Binary format");
```

```
        break;
    }
    printf("\n");
    printf("  Extra Parameters:  %s\n",user_info->extra_
     parameters);
    printf("\n");
    printf (" Alter - Alter standard options\n");
    printf (" Extra - Set extra parameters\n");
    printf ("\n");
    printf (" Load - Load a different object's
            options\n");
    printf (" Save  - Save current object's options\n");
    printf (" Write - Save current option's under
            different object name\n");
    printf (" Copy  - Copy another object's options into
            current object\n");
    printf ("\n");
    printf ("DoSlice - Do slicing with current
            options\n");
    printf (" Quit   - Exit (Use SAVE to save options
            first) \n");
    printf ("\n");
}; /* show_main_menu */

/*******************************************************/

process_UI_main (user_info,str_comm,object_name)
    USER_INFO_STRUC   *user_info;
    char              str_comm[];
    char              object_name[];
{
    unsigned int  option_val,temp;
    char option_str[3],option_chr;
    float temp_float;

    do {
      show_main_menu(user_info,object_name);
```

```
      printf ("Which Command? ");
      getline(option_str,2);
      option_chr = toupper(option_str[0]);
      process_command (option_chr,user_info, str_comm,
       object_name);
    } while (option chr != 'Q');
} /* process_UI_main */

/*****************************************************/

process_command (option_chr,user_info,str_comm,object
 _name)
  char            option_chr;
  USER_INFO_STRUC   *user_info;
  char            str_comm[];
  char            object_name[];
{
  char            temp_name[STR_LEN];

  switch (option_chr)
  {
    case 'A': process_UI_option(user_info,str_comm,
              object_name);
              break;
    case 'E': set_extra_parameters(user_info);
              break;
    case 'L': load_UI_date_file(user_info.object_name);
              break;
    case 'S': save_UI_data_file(user_infor,object_name);
              break;
    case 'W': printf("New object name to save under? ")'
              getline(temp_name,STR_LEN);
              save_UI_data_file(user_info,temp_name);
              break;
    case 'C': copy_UI_data_file(user_info);
              break;
```

```
    case 'D': slice_UI_data_file(user_info,str_comm,
              object_name);
              break;
  }
} /* process_command */

/*******************************************************/
parse_number(line,ptr_passed,num)
  char line[];
  int *ptr_passed;
  float *num;
{
  int sign,ptr;
  float base,fraction_base;
  BOOLEAN good_num;
  ptr = *ptr_passed;

  while (line[ptr] == ' ') ptr++;
  if (line[ptr] == '-') sign = -1;
  else sign = 1;
  if (strchr("+-",line[ptr]) != NULL) ptr++;

  base = 0;
  fraction base = 0;
  good_num = FALSE;
  while ((line[ptr] == '.') || ((line[ptr] >= '0') &&
   (line[ptr] <= '9')))
}
  if (line[ptr] != '.') {
    good_num = TRUE;
    if (fraction_base == 0.0) {
      base = base * 10 + (line[ptr] - '0');
    } else {
      base = base + ((float) (line[ptyr++] - '0' *
fraction
        _base);
      fraction_base = fraction_base / 10.0;
```

```
    }
  } else {
    fraction base = 0.1;
    ptr++;
  }
}

*ptr_passed = ptr;
*num = sign * base;
if (((line[ptr] != ' ') &&(line[ptr] != ',') && (line
  [ptr] != '\0')) || !good
{
  *num = 0;
  return(FALSE);
} else {
  *ptr_passed++;
  return(TRUE);
  }
}
/*******************************************************/
getline(s,lim)
  char s[];
  unsigned int lim;
{
  int c,i;
  FLUSH_IN;
  i = 0;
  c = getchar();
  while ((i<lim && (c != EOF) && (c!='\n') && (c!=LF))
  {
    s[i] = c;
    i++;
    c = getchar();
  }
  s[i] = '\0';
  FLUSH_IN;
  return(i);
```

```
}
/*******************************************************/

find_z_level(user_info,level,index)
  USER_INFO_STRUC   *user_info;
  float     level;
  unsigned  *index;
{
  unsigned i;
  i=0;
  while ((user_info-> z_level[i] < level) &&
         (user_info-> z_level[i] >=0) &&
         (i < MAX_Z_TABLE)) i++;
  *index = i;
}
/*******************************************************/

insert_z_table(user_info,level,spacing,message)
  USER_INFO_STRUC   *user_info;
  float level,spacing;
  char message[];
{
  unsigned i,j;
  if ((user_info-> z_level[MAX_Z_TABLE-1] >= 0.0) &&
      (user_info-> z_level[MAX_Z_TABLE-1] != level))
  {
    strcpy(message,"**** Table Full.  Cannot Add New
     Level ****");
  } else {
    find_z_level(user_info,level,&i);
    if ((user_info-> z_level[i] != level) && (user_
     info-> z_level[i] >= 0))
    {
      for (j=MAX_Z_TABLE-1; (j > i); j--)
      {
        user_info-> z_level[j] = user_info-> z_level
         [j-1];
```

```
            user_info-> z_level_spacing[j] = user_info->
             z_level_spacing[j-1];
          }
        }
        user_info-> z_level[i] = level;
        user_info-> z_level_spacing[i] = spacing;
      }
    }

    /*******************************************************/

    delete_z_level(user_info,level,message)
      USER_INFO_STRUC  *user_info;
      float level;
      char message[];
    {
      unsigned i,j;

      find_z_level(user_info,level,&i);
      if ((user_info-> z_level[i] != level) || (i ==
       MAX_Z_TABLE))
      {
        strcpy(message,"**** Level does not exist ****");
      } else  {
        for (j=i; j<(MAX_Z_TABLE-1); j++)
        {
            user_info-> z_level[j] = user_info-> z_level
             [j+1];
            user_info-> z_level_spacing[j] = user_info->
             z_level_spacing[j+1];
        }
        user_info-> z_level[MAX_Z_TABLE-1] = -1;
      }
    }
```

```
/*******************************************************/

show_z_table(user_info,linenum)
  USER_INFO_STRUC   *user_info;
  unsigned int *linenum;
{
  unsigned int i,next_line;

  CLRSCR;
  printf("      Z Level");
  printf("                                        ");
  printf("Spacing\n");
  printf("      -------");
  printf("                                        ");
  printf("----------\n");
  next_line = 3;
  for (i=0; (user_info->z_level[i] >= 0) && (i < MAX_Z_
   TABLE); i++)
  {
    printf("        %9.3f ",user_info->z_level[i]);
    printf(". . . . . . . . . . . . . . . . .");
    printf(" %9.3f\n",user_info->z_level_spacing[i]);
    next_line++;
  }
  *linenum = next_line;
}
/*******************************************************/

void input_z_spacing (user_info_perm)
  USER_INFO_STRUC  *user_info_perm;
{
  USER_INFO_STRUC user_info;
  char cmd,line[STR_LEN],message[STR_LEN];
  unsigned int i,linenum;
  float float_value,float_value_2;
```

```
CLRSCR;
printf("\n\nLayer Thickness Input\n");
printf("===============\n\n");
do {
  printf("Fixed or Variable Layer Thickness (F, V or
   Q)? ");
  getline(line,1);
  cmd = toupper(line[0]);
} while (strchr(:FVQ",cmd) == NULL);

switch (cmd) {
  case 'F':
    do {
      printf("\nFixed Layer Thickness Value? ");
      getline(line,STR_LEN-1);
      i = 0
      parse_number(line,&i,&float_value);
} while ((float_value <= 0) || (float_value >=
  65535.0));
  user_info_perm-> z_spacing = float_value;
  break;
case 'V':
  strcpy(message," ');
  memcpy(&user_info,user_info_perm,sizeof(USER_INFO
   _STRUC));
  user_info.z_spacing = 0.0;
  do {
    do {
      show _z_table(&user_info,&linenum);
      while (linenum < 23)
      {
        printf("\n");
        linenum++;
      }
      printf("%s\n",message);
      printf("Command (A,D,S,Q,Help)? ");
      getline(line,STR_LEN-1);
```

```
  cmd = toupper(line[0]);
  if (cmd == '\0' || strchr("ADSQH",cmd) == NULL)
    strcpy(message,"**** Enter Valid Comand
     ****");
} while (cmd == '\0' || strchr("ADSQH",cmd) ==
   NULL);
strcpy(message," ")'
switch (cmd) {
  case 'H':
    printf("\nLayer Thickness Help Commands:
     \n\n");
    printf(" A level spacing  Add new level and
                              spacing value to
                              table\n");
    printf(" D level          Delete level from
                              table\n");
    printf(" S                Save table, return
                              to options menu\n");
    printf(" Q                Quit, return to
                              options menu, don't
                              save\n;
    printf("  H               print this Help
                              message\n");
    printf("\n Note: All values are in CAD
           Units\n");
    wait_key();
    break;
   case 'A';
    i = 1;
    if (!parse_number(line,&i,&float_value))
      strcpy(message,"**** Invalid Numeric input
       for level ****");
    else {
      if (float_value < 0.0)
        strcpy(message,"**** Level must be a
         positive number ****");
      else {
```

```
                if (!parse_number(line,&i,&float_value_2))
                  strcpy(message,"**** Invalid Numeric
                   input for spacing ****");
                else {
                  if (float_value_2 <= 0.0)
                    strcpy(message,"**** Spacing must be
                     greater that zero ****");
                  else {
                    insert_z_table(&user_info, float_
                     value, float_value_2,message);
                    }
                }
            }
        }
        break;
      case 'D':
        i = 1;
        if (!parse_number(line,&i,&float_value))
          strcpy(message,"**** Invalid Numeric input for
           level ****");
        else {
          if (float_value < 0.0)
            strcpy(message,"**** Level must be a positive
             number ****");
            else {
              delete_z_level(&user_info, float_value,
               message);
            }
          }
          break;
        case 'S':
          memcpy(user_info_perm,&user_info,sizeof
           (USER_INFO_ STRUC));
          PRINTF("=== Variable Layer Thickness Information
           Saved ===\n");
          wait_key();
          break;
```

```
      } /* ADSQH  */
    } while (strchr("SQ",cmd) == NULL);
    break; /* V */
  } /* VFQ */
}  /* input_z_spacing */

/*****************************************************/

copy_UI_data_file (user_info)
  USER_INFO_STRUC   *user_info;
{
  FILE  *file_ptr;
  char  object_name[STR_LEN],filename[STR_LEN],temp1
    [STR_LEN],temp2[STR_LEN];
  unsigned int file_type;

  printf ("Object name to copy? ");
  getline(object_name,STR_LEN);
  strcpy(filename,object_name);
  strcat (filename,UI_FILE_EXT);
  if (!file_exit(filename))
  {
    printf("**** Cannot find User Interface data file
     for
    %s ****\n",object_name);
  } else {
    strcpy(temp1,user_info-> input_filename);
    file_type = user_info->file_type;
    strcpy(temp2,user_info-> segfile_name);
    get_user_info_from_file(user_info,object_name);
    strcpy(user_info-> input_filename,temp1);
    user_info-> file_type = file_type;
    strcpy(user_info-> segfile_name,temp2);
    printf("User Interface Options for %s Copied\n",
    object_name);
  }
  wait_key();
```

```
} /* copy_UI_data_file */

/*******************************************************/

set_extra_parameters (user_info)
  USER_INFO_STRUC   *user info;
{
  char temp[STR_LEN];

  CLRSCR;
  printf("\n");
  printf("Extra Parameters allows you to specify non-
   standard slice options\n");
  printf("for includsion in the slice command line. What
   parameters are valid\n");
  printf("is based on the version of Slice you will
   use.\n\n");
  printf("The following parameters exist:\n\n");
  printf("  -x        slice part along the x-axis\n");
  printf("  -y        slice part along the y-axis\n");
  printf("  (default) slice part along the z-axis\n");
  printf("\n\n");
  printf("Current Extra Parameters are:\n");
  printf("%s\n\n",user_info->extra_parameters);
  printf("Enter new Extra Parameters or just press Enter
   to keep the old:\n");
  getline(temp,STR_LEN-1);
  if (strlen(temp)>0) {
    if (strcmp(temp," ")==0) strcpy(user_info->extra_
     parameters,"");
    else strcpy(user_info->extra_parameters,temp);
  }
}
/*******************************************************/
load_UI_data_file (user_info,object_name)
  USER_INFO_STRUC   *user_info;
  char              object_name[];
```

```
{
  FILE  *file_ptr;
  char  filename[STR_LEN];

  printf ("Object name to load? ");
  getline(object_name,STR_LEN);
  strcpy(filename,object_name);
  strcat (filename,UI_FILE_EXT);
  if (!file_exist(filename))
  {
    printf("==== Cannot find User Interface data
      file for
     %s\n",object_name);
    printf("==== Setting options to User Interface
     defaults\n");
    init_user_info(user_info,object_name);
  } else {
    get_user_info_from_file(user_info,object_name);
    print("User Interface Options for %s Loaded\n",
     object_name);
  }
  wait_key();
} /* load_UI_data_file */

/*******************************************************/

slice_UI_data_file (user_info,str_comm,object_name)
  USER_INFO_STRUC   *user_info;
  char              str_comm[];
  char              object_name[];
{
  char    z_space_file[STR_LEN],slice_version[5];

  strcpy(z_space_file,object_name);
  strcat(z_space_file,Z_SPACE_EXT);
  printf("Slice Version to use (Default %s)? ",SLICE
   _VER);
```

```
  getline(slice_version,4);
  if (slice_version[0] == '\0') strcpy(slice_version,
    SLICE_VER);
  set_command_line (user_info,object_name,str_comm,
   z_space_file,slice_version);
  run_slice (user_info, str_comm,z_space_file);
} /* slice_UI_data_file */
/*******************************************************/
show_option_menu(user_info,object_name)
  USER_INFO_STRUC   *user_info;
  char        object_name[];
{
  char          filename [STR_LEN];
  unsigned int  i;

  CLRSCR;
  printf ("         - SLICE USER INTERFACE -\n");
  printf ("             Version %-1.2f\n\n\n",VERSION);
  printf ("   Current Object: %s\n\n:,object_name);
  printf ("Option  Description          CAD units);
  printf ("------  -----------          ---------
  printf (" (1)   DATABASE File Name
                              %s ",user_info->in);
  switch (user_info->file_type) {
    case 0:
      printf  ("(ASCII)");
      break;
    case 1:
      printf ("(Binary)");
      break
    }
  printf  ("\n");
  printf  (" (2)  Resolution    '
                                  %-8.3f\n",user_info);
  printf  ("  (3) Layer Thickness
                    "); if (user_info->z_spacing = 0.0)
  {
```

```
  printf ("%-8.3f Fixed %c%u\n", user_info->z_spacing,
   exact(user_info,user_info), scaled(user_info,user_
   info->z_spacing));
} else {
  print ("Variable\n");
}

printf (" (4)   X hatch spacing  %-8.3f  %c%u\n",
  user_info->x_hatch_spacing,exact(user_info,user
   _info->x_hatch_spacing),
  scaled(user_info,user_info->x_hatch_spacing));
printf (" (5)   Y hatch spacing  %-8.3f  %c%u\n",
user_info->y_hatch_spacing,exact(user_info,user
   _info->y_hatch_spacing),
scaled(user_info,user_info->y_hatch_spacing));
printf (" (6) 60/120 degree hatch spacing %-8.3f
  %c%u\n",
user_info->angle_hatch_spacing,exact(user_info,user
   _info->angle_hatch_spacing);
scaled(user_info,user_info->angle_hatch_spacing));
printf (" (7)   X skin fill spacing   %-8.3f
  %c%u\n",
user_info->x_flat_fill_spacing,exact(user_info,user
   _info->x_flat_fill__spacing),
scaled(user_info,user_info->x_flat_fill_spacing));
printf (" (8)   Y skin fill spacing   %-8.3f
  %c%u\n",
user_info->y_flat_fill_spacing,exact(user_info,user
   _info->y_flat_fill__spacing),
scaled(user_info,user_info->y_flat_fill_spacing));
printf (" (9) Minimum Surface Angle for scanned facets
              %u\n",user_info-min_surface_angle);>
printf (" (10) Minimum Hatch Intersect Angle
              %u\n",user_info-min_intersect_angle);>
printf (" (11) Segment Output File Name
              %s\n",user_info->segfile_name);
printf  (" (Q)  Quit, return to Main Menu\n");
```

```
  printf  ("\n");
}; /* show_option_menu */

/*******************************************************/

process_UI_option (user_info,str_comm,object_name)
  USER_INFO_STRUC *user_info;
  char            str_comm[];
  char            object_name[];
{
  unsigned int option_val,temp;
  char option_str[3],option_chr;
  float temp_float;

  do  {
    show_option_menu(user_info,object_name);
    printf ("Alter which Option? ");
    getline(option_str,2);
    option_chr = toupper(option_str[0]);
    option_val = 0;
    temp = 0;
    parse_number(option_str,&temp,temp_float,float);
    option_val = temp_float;
    process_option (option_val,user_info);
  } while (option_chr !=  'Q');
}   /* process_UI_option */

/*******************************************************/

process_option (option_val,user_info)
  int           option_val;
  USER_INFO_STRUC   *user_info;
{
  int   temp;

  switch  (option_val)
  {
```

```
        case 1 :  input_filename (user_info);
                  break;
        case 2 :  printf ("Enter resolution value : ");
                  user info->resolution = read_float(FALSE);
                  if (user_info->resolution == 0)
                    user_info-> resolution = 1.0;
                  break;
        case 3 :  input_z_spacing (user_info);
                  break;
        case 4 :  input_hx (user_info);
                  break;
        case 5 :  input_hy (user_info);
                  break;
        case 6 :  input_ahs (user_info);
                  break;
        case 7 :  input_hfx (user_info);
                  break;
        case 8 :  input_hfy (user_info);
                  break;
        case 9 :  input_msa (user_info);
                  break;
        case 10:  input_mia (user_info);
                  break;
        case 11:  printf ("Enter segment file name : ");
                  getline(user_info->segfile_name,STR_
                  LEN-1);
                  if (strchr(user_info->segfile_name,'.') ==
                  NULL)
                    strcat(user_info->segfile_name,SLICE_
                    OUTPUT_EXT);
                  break;
    }
} /* process_option */
/*****************************************************/
input_filename (user_info)
  USER_INFO_STRUC *user_info;
{
```

```
  char  line[2];
  printf  ("Object Data File Name? ");
  getline(user_info->input_filename,STR)_LEN-1);
  if (strchr(user_info->input_filename,'.') == NULL)
      strcat(user_info->input_filename,SLICE_INPUT_EXT);
  do {
    printf ("File Type Binary or ASCII (B,A)? ");
    getline(line,1);
    line[0] = toupper(line[0]);
  } while ((line[0] != 'A') && (line[0] != 'B'));
  if (line[0] == 'A') user_info->file_type = 0;
  else user_info->file_type = 1;
} /* input_filename */

/*******************************************************/

input_hfy (user_info)
  USER_INFO_STRUC   *user-info;
{
  printf ("Enter Y skin fill spacing value (hfy): ");
  (user_info->y_flat_fill_spacing = read_float(TRUE);
} /* input_hfy */

/*******************************************************/

input_hfx (user_info)
  USER_INFO_STRUC   *user_info;
{
  printf  ("Enter X skin fill spacing value (hfx): ");
  user_info->x_flat_fill_spacing = read_float(TRUE);
} /* input_hfx */

/*******************************************************/

input_hy (user_info)
  USER_INFO_STRUC   *user-info;
{
```

```
  printf  ("Enter Y hatch spacing value (hy):");
  user_info->y_hatch_spacing = read_float(TRUE);
} /* input_hy */

/*******************************************************/

input_hx (user_info)
  USER_INFO_STRUC   *user_info;
{
  printf ("Enter X hatch spacing value (hx): ");
  user_info->x_hatch_spacing = read_float(TRUE);
} /* input_hx */

/*******************************************************/

input_ahs (user_info)
  USER_INFO_STRUC   *user_info;
{
  printf ("Enter a value for 60-120 angle hatch
    spacing : ");
  user_info->angle_hatch_spacing = read_float(TRUE);
} /* input_ahs */

/*******************************************************/

input_mia (user_info)
  USER_INFO_STRUC   *user_info;
{
  printf ("Enter Minimum Intersect Angle value (mia):");
  user_info->min_intersect_angle = read_int();
} /* input_mia */

/*******************************************************/

input_msa (user_info)
  USER_INFO_STRUC   *user_info;
{
```

```
  printf ("Enter a Minimum Surface Angle value
    (msa): ");
  user_info->min_surface_angle = read_int();
} /* input_msa */

/*******************************************************/

select_centering (user_info)
  USER_INFO_STRUC   *user_info;
{
  char  ans;

  printf ("Enter Y for centering, N for not centering
   : ");
  ans = getchar();
  if (toupper(ans) == 'Y') user_info->centering = TRUE;
  else user_info->centering = FALSE;
} /* input_centering */

/*******************************************************/

init_user_info (user_info,object_name)
  USER_INFO_STRUC   *user_info;
  char              object_name[];
{
  unsigned i;

  user_info-> version = VERSION:
  strcpy(user_info->input_filename,obect_name);
  strcat(user_info->input_filename,SLICE_INPUT_EXT);
  user_info->file_type              = 1;
  user_info->resolution             = 1000;
  user_info->z_spacing              = 0.01;
  user_info->y_flat_fill_spacing    = 0.0;
  user_info->x_flat_fill_spacing    = 0.003;
  user_info->y_hatch_spacing        = 0.0;
  user_info->x_hatch_spacing        = 0.0;
```

```
  user_info->angle-hatch_spacing  = 0.05;
  user_info->min_surface_angle      = 50;
  user_info->min_intersect_angle    = 0;

  strcpy(user_info->extra_parameters,"");
  strcpy (user_info->segfile_name,object_name);
  strcat (user_info->segfile_name, SLICE_OUTPUT_EXT);
  user_info->centering = FALSE;

  for (i=0; i < MAX_Z_TABLE; i++)
  user_info-> z_level[i] = -1;
} /* input_user_info */

/*****************************************************/

set__command_line (user_info,object_name,str_comm,z_
 space_file,slice_version);
  USER_INFO_STRUC   *user_info;
  char              object_name[];
  char              str_comm[];
  char              z_space_file[],slice_version[];
{
  char    ch[STR_LEN];
  int     i;
  FILE    *file_ptr;

  strcpy (str_comm, SLICE_PGM);
  strcat (str_comm, slice_version);
  strcat (str_comm, " <");
  strcat (str_comm, user_info->input_filename);
  strcat (str_comm, " ");

  switch (user_info->file_type) {
    case 1:
      strcat  (str_comm, BINARY_TXT);
      strcat  (str_comm, " ");
      break;
```

```
    }

  if (strcmp(slice_version,"240") < 0)
    strcat (str_comm, SCALE_TXT);
  else strcat (str_comm, RES_TXT);
  sprintf (ch, " %-1.3f ", user_info->resolution);
  strcat (str_comm,ch);

  if (user_info->z_spacing != 0.0)
{
  strcat (str_comm, Z_SPACE_TXT);
  sprintf (ch, " %u ", scaled(user_info,user_info->z_
   spacing));
  strcat  (str_comm, ch);
} else  {
  strcat (str_comm, VAR_Z_SPACE_TXT);
  strcat (str_comm, " ");
  strcat (str_comm, z_space_file);
  strcat (str_comm, " ");

  openfile (z_space_file, &file_ptr, "w");
  for (i=0; ((i < MAX_Z_TABLE) && (user_info->z_level[i]
  >= 0)); i++)
    fprintf(file_ptr, "%u %u\n",
      scaled(user_info,user_info->z_level[i]),
      scaled(user_info,user_info->z_level_spacing[i]));
    fclose(file_ptr);
  }
if (user_info->x_flat_fill_spacing != 0)
{
    strcat  (str_comm, X_FLAT_FILL_TXT);
    sprintf (ch, " %u ", scaled(user_info,user_info->
     x_flat_fill_spacing));
    strcat (str_comm, ch);
}
if (user_info->y_flat_fill_spacing != 0)
{
```

```
    strcat  (str_comm, Y_FLAT_FILL_TXT);
    sprintf (ch, " %u ", scaled(user_info,user_ info->
     y_flat_ fill_spacing));
    strcat (str_comm, ch);
  }
  if (user_info->x_hatch_spacing != 0)
  {
    strcat  (str_comm, X_HATCH_TXT);
    sprintf (ch, " %u ", scaled(user_info, user_info->
     x_hatch_spacing));
    strcat (str_comm, ch);
  }
  if (user_info->y_hatch_spacing != 0)
  {
    strcat  (str_comm, Y_HATCH_TXT);
    sprintf (ch, " %u ", scaled(user_info,user_info->
     y_hatch_spacing));
    strcat (str_comm, ch);
  }
  if (user_info->angle_hatch_spacing !=0)
  {
    strcat  (str_comm, ANGLE_HATCH_TXT);
    sprintf (ch, " %u ", scaled(user_info,user_info->angle
      _hatch_spacing));
    strcat  (str_comm, ch);
  }

  if (user_info->min_surface_angle != 0)
  {
    strcat  (strcomm, MSA_TXT);
    sprintf (ch, " %u ", user_info->min_surface_angle);
    strcat  (str_comm, ch);
  }
  if (user_info->min_intersect_angle != 0)
  {
```

```
  strcat  (str_comm, MIA_TXT);
  sprintf (ch,  " %u ", user_info->min_intersect_angle);
  strcat (str_comm, ch);
}
if (user_info->centering)
{
  strcat  (str_comm, CENTERING_TXT);
  strcat  (str_comm, "  ");
}

strcat (str_comm, OUT_FILE_TXT);
strcat (str_comm, " ");
strcat (str_comm, user_info->segfile_name);

if (strlen(user_info->extra_parameters)>0) {
  strcat  (str_comm, " ");
  strcat  (str_comm, user_info->extra_parameters);
}
} /* set_command_line */

/*******************************************************/

run_slice (user_info,str_comm,z_space_file)
  USER_INFO_STRUC   *user_info;
  char              str_comm[];
  char              z_space_file[];
{
  int rc,i = 0,j = 0;
  char temp[STR_LEN];

/*
  printf ("Slice Command Line:\n);
  while (str_comm[i]  != \0') {
    if ((j  > 65) && (str_comm[i] == ' '))  {
        j = 7;
        printf("\n           ");
      }
```

```
      printf  ("%c", str_comm[i++]);
      j++;
    };
    printf("\n\n");
*/
  fflush(stdout);

/*
    timestamp and execute slice command
*/
  time(&system_time);
  time_val1 = times(&timestamp1);
  rc = system (str_comm);
  time_val2 = times(&timestamp2);

/*
    record slice activity in log file
*/
  openfile (log_file_name, &log_file_ptr, "a");
  fprintf(log_file_ptr, "%s\n", str_comm);
  fprintf(log_file_ptr, "%s CODE=%d REAL=%lu CPU=%lu\n",
   ctime(&system_time),rc,
    time_val2-time_val1,
    timestamp2.tms_cutime + timestamp2.tms_cstime
    _timestamp1.tms_cutime - timestamp1.tms_cstime);
  fclose(log_file_ptr);

  if (file_exist(z_space_file))
  {
    strcpy(temp,rm ");
    strcat(temp, z_space_file);
    system(temp);
  }
  if (rc != 0)
  {
    printf  ("\n*\n);
```

```
      printf  ("* WARNING:  SLICE ERROR DETECTED RC =
      %d\n," rc);
      printf ("*\n");
    } else printf (" SLICE COMPLETED\n");

    wait_key();
  } /* run_slice */
  /*******************************************************/

  main(argc,argv)
    int  argc;
    char *argv[];
  {
    USER_INFO_STRUC   user_info;
    char        object_name[STR_LEN];
    char        str_comm[200], ans[STR_LEN],ans2 [STRLEN];

    strcpy  (ans, "      ");
    get_object_name (argc,argv,object_name);
    get_user_info_from_file (&user_info, object_name);

    process_UI_main (&user_info,str_comm,object_name);
    update_last_UI (object_name);
    printf("\n");
    exit(0);
  }
```

SLICE.H

```
#define   LF                  10
#define   MAX_Z_TABLE         20
#define   STR_LEN             81
#define   BOOLEAN             short int
#define   TRUE                1
#define   FALSE               0

#define   VERSION_TXT         "version"
#define   IN_FILE_TXT         "datafile"
#define   FILETYPE_TXT        "filetype"
#define   EXTRA_PARAMS_TXT    "extra"

#define   OUT_FILE_TXT        "-sf"
#define   Z_SPACE_TXT         "-zs"
#define   SCALE_TXT           "-scale"
#define   RES_TXT             "-res"
#define   X_FLAT_FILL_TXT     "-hfx"
#define   Y_FLAT_FILL_TXT     "-hfy"
#define   X_HATCH_TXT         "-hx"
#define   Y_HATCH_TXT         "-hy"
#define   ANGLE_HATCH_TXT     "-ha"
#define   MSA_TXT             "-msa"
#define   MIA_TXT             "-mia"
#define   BINARY_TXT          "-b"
#define   CENTERING_TXT       "-c"
#define   VAR_Z_SPACE_TXT     "-zsf"

typeoef struct      /* NOTE: Not the same as in SLICE */
{
     float     version;
     char      input_filename[STR_LEN];
     unsigned  file_type;
     char      segfile_name[STR_LEN];
     float     resolution;
     float     z_spacing;
```

```
        float     x_flat_fill_spacing;
        float     y_flat_fill_spacing;
        float     x_hatch-spacing;
        float     y_hatch_spacing;
        float     angle_hatch_spacing;
        unsigned  min_surface_angle;
        unsigned  min_intersect_angle;
        unsigned  centering;
        char      extra_parameters[STR_LEN];
        float
z_level[MAX_Z_TABLE],z_level_spacing[MAX_Z_TABLE];
} USER_INFO_STRUC;
```

```
STEREO.PAS
{=======================================================
     STEREO, Sterolithography Device Driver
     for 3D Systems' Sterolithography System
     Version 3.00 by Ternz Technologies

=======================================================

STEREO is a memory-resident laser controller device
driver.  It interfaces between the Background Task's
Software Interrupt Handler and a program wanting to
control the laser apparatus using easy text-based
commands.

STEREO receives and processes simple (X,Y) laser
position commands passed to it by some other program
using a software interrupt vector.  The CLIFF Turbo
Pascal program is just such a program; it can read
commands from either the computer's keyboard or from
disk files and then pass them along to this program.

STEREO, like an MSDOS driver, can be called by programs
written in any language, including assembly and BASIC.
The interface will usually be a special subroutine that
invokes the STEREO software interrupt.  Unlike an MSDOS
driver, STEREO cannot be referenced directly by a
filename.

STEREO's device driver format is simpler, faster, and
more clean than an MSDOS device driver.  STEREO
processed a complete text string (a line) at a time, as
opposed to one character at a time for an MSDOS device
driver.  If desired, an official MSDOS device driver
(STL:, LSR: or whatever) could be added in later.
```

STEREO understands the following commands:

	BX	0-65535	Beginning X position	(position)
	BY	0-65535	Beginning Y position	(position)
	JX	0-65535	Jump to new X position	(position)
	JY	0-65535	Jump to new Y position	(position)
	NX	0-65535	Next X position	(position)
	NY	0-65535	Next Y position	(position)
	SS	1-65535	Step Size	(rel incr)
!	SP	5-6553,..	multi-pass Step Periods	(10 us)
	SD	0-65535	Scanning Delay	(step periods)
	JS	1-65535	Jump Size	(rel incr)
	JD	0-65535	Jump Delay	(step periods)
	LO	0-65535	Laser On delay	(10 us)
	LF	0-65535	Laser Off delay	(10 us)
+	NS		eNter Sequence mode	-
+	AB		Enter Absolute mode	-
	EX		Execute	-
	EC		Execute and Clear	-
	CL		Clear table	-
!	MD	AS	+/-set Auto-Shutter mode	(on/off)
!	MD	BL	+/-set BLock mode	(on/off)
!	MD	PA	+/-set Pos Ack mode	(on/off)
!	SR		Soft Reset	
!	HR		Hard Reset	
!	RC	1-10	Redraw pass Court	(# passes)
!	RD	0-65335	Redraw Delay	(step periods)
!	RS	0-65535	Redraw Size	(distance)
!	VC	1-10	riVet pass Count	(# passes)
!	VP	5-6553,..	riVet step Periods	(10 us)
!	VR	0-65535	riVet Reduction amount	(distance)
!	VX	0-65535	riVet end X position	(position)
!	VY	0-65535	riVet end Y position	(position)
!	WI	0-32767	Wait for Idle	(delay)

- New Version 2.80 Commands -

```
!   DC   +/-  set Drift Correction mode (on/off)
!   R1   0-65535,0-65535 new sensor
         #1 Reference (X,Y)  position
!   R2   0-65335,0-65535 new sensor
         #2 Reference (X,Y) position
!   D1   0-65535,0-65535 new sensor
         #1 Drift adjustment (X,Y) position
!   D2   0-65535,0-65535 new sensor
         #2 Drift adjustment (X,Y) position
!   BG   0-65535,0-65535 define BeGin
         (X,Y) position
!   JP   0-65535,0-65535 JumP to new
         (X,Y) position
         ( preferred over JX, JY combination )
!   MV   0-65535,0-65535 MoVe to next
         (X,Y) position
         ( preferred over NX, NY combination )
!   RV   0-65535,0-65535 make a RiVet to next
         (X,Y) position
         ( preferred over VX, VY combination )
```

where:

10 us	is the # of 10s cf microseconds resolution = 0.84 us (from 1.19 MHz clock)
position	is the absolute position in the field
distance	is the absolute distance in step size units
rel incr	is the relative # of position increments
# passes	is the number of redraw passes
step periods	is the time to cover that # of steps programmed by SP command (= tick counts)
delay	delay value in milliseconds (may be faster on faster computers)

In the above list, following each two-character command is the range of numbers allowed as arguments to the command. Each command must be on a separate line, ending with a CR or CR/LF sequence. All spaces are ignored. All numbers are in unsigned decimal.

The Control Instruction Set is intended to be compatible with General Scanning's DG Series Electronic Packages for Laser Control. Only the Vector Mode is supported. The Raster Mode is not supported. The step and delta functions of the Vector Mode are also unsupported. No Z-axis control is supported yet. Commands marked with an '*' are not fully implemented as per the DG specification for functional reasons. Commands marked with '+' are included only for the sake of complete command compatibility. '!' denotes new commands not available on the DG devices.

DG Version 2.6 Vector Mode Commands not supported:

ST	enter STep mode
RE	REset to top of stack in step mode
DL	enter DeLta mode
Q1	Query IB 1
Q2	Query IB 2

The command line with invokes STEREO and causes it to be installed in memory takes the form:

STEREO [/options]

where /options are any number of user-selectable options:

/D	-	debug mode, display vector information
/NV	-	no constant vector velocity
/NG	-	no geometric correction
/NC	-	no version check at install time

STEREO can only be installed once after the computer has been powered up. If different options are required, the computer system will have to be completely rebooted.

The SP (and VP) command has multiple arguments. Each of up to ten passes of redraw can have its own step period. The SP command can have 1 to 10 arguments, each a step period for a pass:

SP sp0,sp1,sp2,sp3...

where sp0 is the step period for redraw pass 0 (the original vector move), sp1 is the step period for redraw pass 1, and so on. /V121 option is not longer supported.

Geometric correction is automatically invoked. If geometric correction is not desired, the option /NG or /GEO NO should be specified on the command line. Geometric correction occurs on the endpoints of any vector STEREO passes to the software interrupt handler.

STEREO code came from Version 1.40 of CLIFF, which originally performed the command translation. This memory-resident STEREO device driver was formed to speed up 3D Systems' supervisory programs, written in GWBASIC and Turbo BASIC. With the Geo Correct Table, STEREO takes up approximately 80K of memory, increasing top about 150K the amount of main memory used by the complete laser controller system and its data structures.

Axes Control Options:		DEFAULT
/X +/-	select X-axis direction	+
/Y +/-	select Y-axis direction	+
/SWAPXY	swap X and Y coordinates	no
/GEO XY/YX/NO	select geo correct sequence (used for old style)	YX
/AXESBEGIN	to axes-control begin pos	no

==========

Recent History:

8/18/87	Ver 2.02	STEREO released
		memory resident portion of Ver 1.x CLIFF process WI
		Wait for Idle command added
8/22/87	Ver 2.03	Stuart's new Geometric Correction
		Y is the first, X is the second mirror
		/NC no version check flag
		jump vectors of 0 steps skipped
		CvtInt now works for 32768 ($8000)

		Step periods supported for stream (non-block) mode
8/25/87	Ver 2.04	Geometric Correction delta bugs fixed
9/23/87	Ver 2.12	STEREO Axes Control added new options /X, /Y, /SWAPXY, /GEO
9/30/87	Ver 2.25	Table-Interpolation Geometric Correction installed
10/26/87	Ver 2.26	Wait for Idle argument is 100 minimum /NG prevents loading of STEREO.GEO
10/28/87	Ver 2.27	improved Geometric Correction formula
11/12/87	Ver 2.30	first Beta software release
11/18/87	Ver 2.31	allow axes control independent of geo correction /AXESBEGIN option added to axes-control BX,BY GeoCorrectTable size made programmable (70 elements/axis) added TableDivisor for GeoCorrectTable
12/11/87	Ver 2.40	STEREO upgraded to Turbo Pascal Version 4.0 STEREO always returns with Queue Size Change in CX GeoCorrectTable storage allocated on heap dynamically max GeoCorrectTable size increased to 90 by 90 Some variables changed to Word or LongInt type
12/16/87	Ver 2.41	GeoCorrect overwrite / memory alloc debugged

```
                        QueueCount substituted for QueueSize
                        calculation
        12/21/87  Ver 2.43  Check for Break turned off
                        improved keyboard and Ctrl-Break
                        Abort service
        1/06/68   Ver 2.50  zero-length vectors ignored
                        Ctrl-Break check looks at scan code
                        now
        1/27/89   Ver 2.60  better STEREO.GEO processing
                        blanks allowed around commas
                        separating numbers
                        check for too few or too many
                        entries in STEREO.GEO no mark and
                        redraw around executes when redraw
                        count is 1
                        OkayForRedraw Command flag added to
                        limit redundant mark and redraw
                        commands
        3/09/88   Ver 2.80  added copyright notice
                        implemented drift correction logic
                        for profiler sensors
                        reduced ASCII overhead for begin,
                        jump, and next commands
                        installed support for binary STEREO
                        commands
                        new STEREO commands:  DC, R1, R2,
                        D1, D2, BG, JP, MV, RV
        4/09/88   Ver 3.00  production release

        }
        {$R-,S-,I-,D+,T-,F-,V-,B-,N-,L+}
        {$M 4096,1,65536}

        Uses Dos,Crt;
```

```
type
     str4  =   string[4];
     str16 =   string[16];
     str80 =   string[80];
     BitArray = array[0..15] of Word;
     GeoCorrectTableType = array[0..10000] of LongInt;
     GeoCorrectTablePtr =  GeoCorrectTableType;
     FpPointerStructType = array[0..8] of Byte;
     FpPointerType =  FpPointerStructType;
     PositionType = record
       X: Real;
       Y: Real;

     end;
     SensorDataType = record
       Ref: PositionType;
       Adj: PositionType;
     end;

var
     FirstVar: Byte; { must be first variable defined }
     ch,key: Char;
     temp,cmd,TextCommand,OperationMnemonics,Operand:
String;
     StereoVersion: str16;
     count,i,j,remline,Er,OpNum,code,code2: Integer;
     SensorNum,IOerr: Integer;
     GeoFile,DefaultFile: text;
     tempX,tempY,X1,Y1,X2,Y2,oldX2,oldY2: Real;
     arg,beginXpos,beginYpos,Xpos,Ypos,oldX,oldY: Real;
     FirstX,NegX,NegY,SwapXY,SwapBegin: Boolean;
     first,VelocityFlag,DebugMode,Redrawing,done:
Boolean;
     OkayForRedrawCommand: Boolean;
     _Step_Size,_Step_Period,_Scan_Delay,_Jump_Delay:
Word;
     _Jump_Size,_Rivet_Reduction: Real;
```

```
      _Laser_on_Delay,_Laser_Off_Delay: Word;
      _Redraw_Count,_Redraw_Delay,_Redraw_Size: Word;

    KeyAbort,VersionCheckFlag,GeoCorrectFlag,DoingRivet:
    Boolean;
      DriftCorrectionFlag,BinaryFlag: Boolean;
      changeX,changeY,ChangeV,BrokenX,BrokenY: Real;
      deltaX,deltaY,steps,GeoDeltaX,GeoDeltaY: Real;
      Distance,IntDeltaX,IntDeltaY: Integer;
      IntSteps,TotalSteps,TempSteps: Word;
      RivetMainPeriod,CurrentRedrawCount,_Rivet_Count:
    Word;
      StoreCount,QueueCount: Word;
      angX,angY,tanX,GeoScale,GeoFact: Real;

    FieldX,FieldY,StartX,StartY,EndX,EndY,GeoEndX,GeoEndY,Xc
    oord,Ycoord: Real;
      DiscreteX,DiscreteY,DiscreteSteps: Word;
      regs: Registers;
      XC,YC,XCount,YCount,CoordCount,TableDivisor: Word;
      StereoSize: LongInt;
      CommandOfs,CommandSeg: Word;
      data,Return: Byte;
      XGeoCorrectTable,YGeoCorrectTable:
    GeoCorrectTablePtr;
      Sensor: array[1..2] of SensorDataType;
      DriftGain: PositionType;
      LastVar: Byte;   { must be last variable defined }

    const
      ThisStereoVersion = '3.00';
      MinSystemVersion  = '2.30';
      CtrlBreak = #0;
      HexDigits: str16 = '0123456789ABCDEF';
                                  sp40 = '
      Indent1 = 4;
      Indent2 = 41;
```

```
    AllBits = $FFFF;
    BitValue: Bit Array =
       ($0001, $0002, $0004, $0008, $0010, $0020,
$0040, $0080,
        $0100, $0200, $0400, $0800, $1000, $2000,
$4000, $8000);
    Shutter = 0; { control word bit 0 - shutter
open/closed }
    _BIOSVECTOR   = $16;
    _STEREOVECTOR = $65;
    MaxGec = 100;       { maximum GeoCorrectTable Index
}

    GeoMult = 4096;     { GeoCorrectTable LongInt
multiplier }
    StepRound = 0.02;   { rounding value for # of laser
steps }

label ExitProgram;

function StartOfProgramCode: String;
begin
  StartOfProgramCode :=
     'STEREO Device Driver Version 2.80 Copyright (C)
1988 by Tarnz Technologies';
end;

{$i laser.inc} { include laser interface library }

var
  StepPeriodsArray,RivetPeriodsArray: _StepPeriodsType;
function_StereoOK: Boolean;
type
  CopyType = array[1..20] of char;
var
  CopyPtr:  ^CopyType;
  CopyStr:  string[20];
```

```
begin
  with regs do begin
    AX := ($35 shl S) or _STEREOVECTOR;
    MsDos(regs);
    CopyPtr := Ptr(ES,57);
  end;
  move(CopyPtr^[1],CopyStr[1],18);
  CopyStr[0] := #18;
  if CopyStr = 'Tarnz Technologies' then begin
    __StereoOk := true;
    with regs do
      CopyPtr := Ptr(ES,30);
    move(CopyPtr^[1],StereoVersion[1],4);
    StereoVersion[0] := #4;
  end else begin
    __StereoOk := false;
    StereoVersion := '?,??';
  end;
end;

function upper(text: String): String;
var
  i: Integer;
  temp: String;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer); String;
var
  temp: String;
  i: Integer;
begin
  temp := '';
```

```
  for i:=1 to num do temp :- temp+Ch;
  many := temp;

end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

function CvtDelta(num: Real): Word;
begin
  num := num * 256;
  if num<32768.0 then CvtDelta := trunc(num)
    else if num>32768.0 then CvtDelta := trunc(num)
      else CvtDelta := $8000;
end;

function sgn(num: Real): Integer;
begin
  if num  0 then sgn := -1
    else if num>0 then sgn := 1
      else sgn := 0;
end;

function CvtTicks(time: Real): Word;
begin
  CvtTicks := trunc(time * 119.0 / 10.0);
end;

procedure SkipToNum(var str: String);
var
  i: Integer;
```

```
begin
  i := 1;
  while (i< = length(str)) and not(str[i] in
['0'..'9','.']) do inc(i);
  if i>1 then str := copy(str,i,99);
end;

procedure RealParse(var RealNum: Real);
var
  i: Integer;
begin
  i := 1;
  while (i <= length(temp)) and (temp[i] = ' ') do
inc(i);
  delete(temp,1,i-1);
  if copy(temp,1,1) = '.' then insert('0',temp,1);
  val(temp,RealNum,code);
  if code <> 0 then val(copy(temp,1,code-1),
     RealNum,code2);
  temp := copy(temp,code+1,99);
end;

procedure IntParse(var IntNum: Word);
var
  num: Real;
begin
  if length(temp)=0 then IntNum := 0
  else begin
    val (temp,num,code);

  if code <> 0 then val(copy(temp,1,code-1),num,code2);
  IntNum := trunc(num);
  if code=0 then temp := ''
    else temp := copy(temp,code+1,99);
 end;
end;
```

```
function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num >=0 then Str(num,temp) else
Str(num+$8000,temp);
  IntStr := temp;
end;

function RealStr(num: Real): str16;
var
  temp: str16;
begin
  if num <0 then Str(65536.0+num:6:0,temp)
    else Str(num:6:0,temp);
  while temp[1]=' ' do temp := copy(temp,2,6);
  RealStr := temp;
end;

function spc(len: Integer): str80;
begin
  spc := copy(sp40+sp40+sp40,1,len);
end;

procedure msg(text: String);
begin
(* Return text to calling program *)
end;

procedure msgln(text: String);
begin
(* return text + Cr to calling program *)
end;

function MemAbs(pt: Pointer): LongInt;
begin
  MemAbs := LongInt(seg(pt^)) shl 4 + ofs(pt^);
```

```
end;

{
  converts a Single type (4 bytes) number to a Real (6
bytes)
}
procedure ConvertStoR(s,r: FpPointerType);
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;

function CharArgument: Char;

begin
  if BinaryFlag then begin
    CharArgument := char(Mem[CommandSeg:CommandOfs]);
    inc(CommandOfs);
  end else begin
    if length(Operand)>0 then CharArgument := Operand[1]
      else CharArgument := #0;
    delete(Operand,1,1);  { slow, avoid for long strings
}
  end;
end;

function WordArgument: Word;
var
  WordArg: Word;
  code,code2: Integer;
begin
  if BinaryFlag then begin
```

```
    WordArgument := Mem[CommandSeg:CommandOfs] shl 8 +
                    Mem[CommandSeg:CommandOfs + 1];
    inc(CommandOfs,2);
  end else begin
    val(Operand,WordArg,code);
    if code <> 0 then val(Operand,WordArg,code2);
    WordArgument := WordArg;
    if code = 0 then Operand := '
      else delete(Operand,1,code);
  end;
end;

function RealArgument: Real;
var
  RealArg: Real;
begin
  if BinaryFlag then begin
    ConvertStoR(@Mem[CommandSeg:CommandOfs],@RealArg);
    RealArgument := RealArg;
    inc(CommandOfs,4);
  end else begin
    if copy(Operand,1,1) = '.' then
insert('0',Operand,1);
    val(Operand,RealArg,code);
    if code <> 0 then val(copy(Operand,1,code-
1),RealArg,code2);
    RealArgument := RealArg;
    if code = 0 then Operand := ' '
      else delete(Operand,1,code);
  end;
end;

function TimeArgument: Word;
begin
  TimeArgument := CvtTicks(WordArgument);
end;
```

```
function PositionArgument: Word;
var
  PosArg: Word;
  code,code2: Integer;
begin
  if BinaryFlag then begin
    PositionArgument := MemW[CommandSeg:CommandOfs];
    inc(CommandOfs,2);
    end else begin
      while copy(Operand,1,1) = ' ' do
delete(Operand,1,1);
      val(Operand,PosArg,code);
      if code <> 0 then val(copy(Operand,1,code-
1),PosArg,code2);
      PositionArgument := PosArg;
      if code = 0 then Operand := ''
        else delete(Operand,1,code);
     end;
end;

function PtrAdd(point: Pointer;offset: LongInt):
Pointer;
var
  NewAddr: LongInt;
begin
  NewAddr := MemAbs(point)+offset;
  PtrAdd := Ptr(NewAddr div 16,NewAddr mod 16);
end;

function StatusKey: Char;
begin
  with regs do begin
    AX := $0100;
    Intr(_BIOSVECTOR,regs);
    Statuskey := char(AH); { return scan code }
  end;
end;
```

```
procedure SIHerror;
begin
  Return := Er;
end;

function TryCommand: Boolean;
begin
  if Er=3 (queue full) then TryCommand := true
    else TryCommand := false;
  if keypressed {abort} then begin
    if Statuskey = CtrlBreak then begin
      TryCommand := false;
      KeyAbort := true;
    end;
  end;
end;

function TryBlockCommand: Boolean;
begin
  if (Er=3) {command queue full} or (Er=6) {exec queue
full}
    then TryBlockCommand := true
      else TryBlockCommand := false;
  if keypressed {abort} then begin
    if StatusKey = CtrlBreak then begin
      TryBlockCommand := false;
      KeyAbort := true;
    end;
  end;
end;

procedure newX(num: Real);
begin
  oldX := Xpos;
  Xpos := num;
end;
```

```
procedure newY(num: Real);
begin
  oldY := Ypos;
  Ypos := num;
end;

{
    AxesControl -- adjust axes
}
procedure AxesControl(oldX,oldY: Real;var retX,retY:
Real);
begin
  if SwapXY then begin
    retY := oldX;
    retX := oldY;
  end else begin
    retX := oldX;
    retY := oldY;
  end;
  if NegX then retX := 65535.0 - retX;
  if NegY then retY := 65535.0 - retY;
end;

procedure SetRedrawCount(RedrawCount: Integer);
begin
  if RedrawCount <> CurrentRedrawCount then begin
   CurrentRedrawCount := RedrawCount;
    Er := 3;
    while TryCommand do
      Er := _SetRedrawCount(CurrentRedrawCount);
    inc(QueueCount,3);
    SIHerror;
  end;
end;
```

```
{
   Laser Motion Routines:

   * ResetRedraw     - just mark for beginning of block
   * PerformRedraw   - redraw and mark last section
   * MoveLaser       - new move alg using redraw & vector
                       vector slicing
   * JumpLaser       - controlled breakup of jump motion
                       to limit overshoot

procedure ResetRedraw;
begin
     if OkayForRedrawCommand and _BlockModeOn then begin
          if DebugMode then msgln('Mark '+IntStr
          (_Redraw_Size));
          Er := 3;
          while TryCommand do
               Er := _Mark;
          inc(QueueCount);
          SIHerror;
     end;
     Distance := 0;
     Redrawing := false;
     OkayForRedrawCommand := false;
end;

procedure PerformRedraw;
begin
     if OkayForRedrawCommand and _BlockModeOn then begin
          if DebugMode then msgln('Redraw
'+IntStr(_Redraw_Size));
          Er :=3;
          while TryCommand do
               Er :=_Redraw;
          inc(QueueCount);
          SIHerror;
     end;
```

```
        Distance := 0;
        Redrawing := false;
        OkayForRedrawCommand := false;
end;

{
        CorrectForDrift -- devleop drift gain terms
}
procedure CorrectForDrift;
var
        quotient: Real;
begin
        quotient := Sensor[2].Ref.X - Sensor[1].Ref.X;
        if quotient<> 0 the
                DriftGain.X := (Sensor[2].Adj.X -
Sensor[1].Adj.X) / quotient
                else DriftGain.X := 0;

(*
writeln('CorrectForDrift-X:');
writeln('2Xadj=',Sensor[2].Adj.X:8:2.' -
1Xadj=',Sensor[1].Adj.X:8:2,' /');
writeln('2Xref=',Sensor[2].Ref.X:8:2,' -
1Xref=',Sensor[1].Ref.X:8:2);
writeln('DriftGainX=',DriftGain.X:8:2);
*)
        quotient := Sensor[2].Ref.Y - Sensor[1].Ref.Y;
        if quotient <> 0 then
                DriftGain.Y := (Sensor[2].Adj.Y -
Sensor[1].Adj.Y) / quotient
                else DriftGain.Y := 0;
(*
        writtenln('CorrectForDrift-Y:');
writeln('2Yadj=',Sensor[2].Adj.X:8:2,' -
1Yadj=',Sensor[1].Adj.Y;8:2,' /');
writeln('2Yref=',Sensor[2].Ref.Y:8:2,' -
1Yref=',Sensor[1].Ref.Y:8:2);
```

```
writeln('DriftGainY=',DriftGain.Y:8:2);
*)
end;

{
     GeoCorrect -- geometric correct coordiante pair
}
procedure GeoCorrect(oldX,oldY: Real;var newX,newY:
Real);
var
     GX11,GX12,GX21,GX22,GY11,GY12,GY21,GY22: Real;
     Xf,Yf,XfYf: Real;
     XIndex,YIndex,Lookup: Word;
begin
     if GeoCorrectFlag then begin
          XIndex := trunc(oldX / TableDivisor);
          if XIndex >= XCount then XIndex := XCount-1;
          YIndex := trunc(oldY / TableDivisor);
          if YIndex >= YCount then YIndex := YCount-1;

     Lookup := XIndex * YCount + YIndex;
     GX11 := XGeoCorrectTable^[Lookup];
     GX12 := XGeoCorrectTable^[Lookup+1];
     GX21 := XGeoCorrectTable^[Lookup+YCount];
     GX22 := XGeoCorrectTable^[Lookup+YCount+1];
     GY11 := YGeoCorrectTable^[Lookup];
     GY12 := YGeoCorrectTable^[Lookup+1];
     GY21 := YGeoCorrectTable^[Lookup+YCount];
     GY22 := YGeoCorrectTable^[Lookup+YCount+1];
     Xf   := (oldX - TableDivisor * 1.0 * XIndex) /
             TableDivisor;
     Yf   := (oldY - TableDivisor * 1.0 * YIndex) /
             TableDivisor;
     XfYf := Xf * Yf;
     newX := (GX11 + Xf * (GX21 - GX11) + Yf * (GX12 -
             GX11)
       + XfYf * (GX22 + GX11 - GX12 - GX21)) / GeoMult;
```

```
      newY := (GY11 + Xf * (GY21 - GY11) + Yf * (GY12 -
              GY11)
        + XfYf * (GY22 + GY11 - GY12 - GY21)) / GeoMult;
    end else begin
      newX := oldX;
      newY := oldY;
    end;
    if DriftCorrectionFlag then begin
(*
      writeln('DC-old: ',newX:8:2,',',newY:8:2);
*)
      newX := (newX - Sensor[1].Ref.X) * DriftGain.X +
Sensor[1].Adj.X;
      newY := (newY - Sensor[1].Ref.Y) * DriftGain.X +
Sensor[1].Adj.Y;
(*
      writeln('DC-new: ',newX:8:2,',',newY:8:2);
*)
    end;
    if DebugMode then begin
      writeln('GX11=',GX11/GeoMult:7:2,'
GX12=',GX12/GeoMult:7:2,
        ' GX21=',GX21/GeoMult:7:2,'
GX22=',GX22/GeoMult:7:2);
      writeln('GY11=',GY11/GeoMult:7:2,'
GY12=',GY12/GeoMult:7:2,
        ' GY21=',GY21/GeoMult:7:2,'
GY22=',GY22/GeoMult:7:2);
      writeln('oldX=',oldX:7:2,' oldY=',oldY:7:2,'
newX=',newX:7:2,' newY=',newY:7:2);
      writeln;
    end;
end;
```

```
procedure JumpLaser;
begin
  AxesControl(Xpos,Ypos,EndX,EndY);
  AxesControl(oldX,oldY,StartX,StartY);
  changeX := EndX-StartX;
  changeY := EndY-StartY;
  if VelocityFlag then changeV :=
sqrt(sqr(changeX)+sqr(changeY))
     else if abs(changeX) < abs(changeY) then ChangeV :=
abs(changeY)
          else ChangeV := abs(changeX);
  steps := ChangeV / _Jump_Size;
  if steps>0 then begin
     deltaX := changeX / steps;
     deltaY := changeY / steps;
     IntSteps  := trunc(steps+StepRound) + 1;
     TotalSteps := 1;
     repeat
       BrokenX := StartX + deltaX*TotalSteps;
       BrokenY := StartY + deltaY*TotalSteps;
       TotalSteps := TotalSteps + 1;
       if TotalSteps >= IntSteps then begin
          BrokenX := EndX;
          BrokenY := EndY;
       end;
       GeoCorrect(BrokenX,BrokenY,FieldX,FieldY);
       Er := 3;                                    { add
discrete jump }
       while TryCommand do begin
          Er := __AddVector(trunc(FieldX),trunc(FieldY),
                    0,0,1);                        { incl a 1
tick pause }
          OkayForRedrawCommand := true;
       end;
       inc(QueueCount,13);
       SIHerror;
     until TotalSteps >= IntSteps;
```

```
  end;
end;

procedure MoveLaser;
begin
  AxesControl(Xpos,Ypos,EndX,EndY);
  AxesControl(oldX,oldY,StartX,StartY);
  changeX := EndX - StartX;
  changeY := EndY - StartY;
  if VelocityFlag then changeV := sqrt(sqr(changeX) +
sqr(changeY))
     else if abs(changeX) < abs(changeY) then ChangeV :=
abs(changeY)
          else ChangeV := abs(changeX);
  steps := ChangeV / _Step_Size;
  if steps>=1 then begin
     deltaX := changeX / steps;
     deltaY := changeY / steps;
     IntSteps  := trunc(steps+StepRound);
     IntDeltaX := CvtDelta(deltaX);
     IntDeltaY := CvtDelta(deltaY);
     TotalSteps := 0;
     repeat
       BrokenX := StartX + deltaX * TotalSteps;
       BrokenY := StartY + deltaY * TotalSteps;
       TempSteps := IntSteps - TotalSteps;
       if TempSteps>256 then TempSteps := 256;
       if (CurrentRedrawCount > 1) and (_Redraw_Size>0)
then
          if Distance+TempSteps >= _Redraw_Size then
begin
            TempSteps := _Redraw_Size - Distance;
            Redrawing := true;
          end;
       TotalSteps := TotalSteps + TempSteps;
       Distance := Distance + TempSteps;
       GeoCorrect(BrokenX,BrokenY,FieldX,FieldY);
```

```
{ Geometric Correction addition }
      if GeoCorrectFlag then begin
         GeoCorrect(BrokenX+TempSteps*DeltaX,BrokenY+
         TempSteps*DeltaY,
            GeoEndX,GeoEndY);
         changeX := GeoEndX - FieldX;
         changeY := GeoEndY - FieldY;
         ChangeV := sqrt(sqr(changeX) + sqr(changeY));
         steps := ChangeV / _Step_Size;
         GeoDeltaX := ChangeX / steps;
         GeoDeltaY := ChangeY / steps;
         DiscreteX := CvtDelta(GeoDeltaX);
         DiscreteY := CvtDelta(GeoDeltaY);
         DiscreteSteps := trunc(steps+StepRound);
      end else begin
         DiscreteX := IntDeltaX;
         DiscreteY := IntDeltaY;
         DiscreteSteps := TempSteps;
      end;

      Er := 3;  { send out move command }
      while TryCommand do begin
         Er := __AddVector(trunc(FieldX),trunc(FieldY),
                  DiscreteX,DiscreteY,DiscreteSteps);
         OkayForRedrawCommand := true;
      end;
      inc(QueueCount,13);
      SIHerror;
      if Redrawing then PerformRedraw;
    until TotalSteps = IntSteps;
  end;
end;

{
   - - - - -
}
```

```
procedure BeginX;
begin
  beginXpos := PositionArgument;
end;

procedure BeginY;
var
  X1,Y1,X2,Y2: Real;
begin
  beginYpos := PositionArgument;
  if SwapBegin then
AxesControl(beginXpos,beginYpos,X1,Y1)
     else begin
       X1 := beginXpos;
       Y1 := beginYpos;
     end;
  GeoCorrect(X1,Y1,X2,Y2);
  Er := __SetHomePosition(trunc(X2),trunc(Y2));
  SIHerror;
end;

procedure BeginPosition;
begin
  BeginX;
  BeginY;
end;

procedure JumpX;
begin
  newX(PositionArgument);
end;

procedure JumpY;
begin
  newY(PositionArgument);
  if _Jump_Delay<>0 then begin
     JumpLaser;
```

```
      Er := 3;
      while TryCommand do
        Er := __AddPause(_Jump_Delay);
      inc(QueueCount,3);
      SIHerror;
    end;
  end;

  procedure PositionJump(X,Y: Real);
  begin
    NewX(X);
    NewY(Y);
    if _Jump_Delay<>0 then begin
      JumpLaser;
      Er := 3;
      while TryCommand do
        Er := __AddPause(_Jump_Delay);
      inc(QueueCount,3);
      SIHerror;
    end;
  end;

  procedure JumpPosition;
  begin
    JumpX;
    JumpY;
  end;

  procedure NextX;
  begin
    newX(PositionArgument);
  end;

  procedure NextY;
  begin
    newY(PositionArgument);
    MoveLaser;
```

```
end;

procedure PositionMove(X,Y: Real);
begin
  newX(X);
  newY(Y);
  MoveLaser;
end;

procedure MoveToPosition;
begin
  NextX;
  NextY;
end;

procedure StepSize;
begin
  _Step_Size := WordArgument;
end;

procedure StepPeriods;
var
  i: Integer;
begin
  PerformRedraw;
  if BinaryFlag then begin
     count := WordArgument;
     for i:=0 to count-1 do StepPeriodsArray.periods[i]
:= TimeArgument;
  end else begin
     count := 0;
     while (Operand <> '') and (count < 10) do begin
       StepPeriodsArray.periods[count] := TimeArgument;
       inc(count);
     end;
  end;
  StepPeriodsArray.num := count;
```

```
  if count>0 then begin
(*   if DebugMode then begin
       msg('New step periods array: ');
       for i:=0 to count-1 do
msg(StepPeriodsArray.periods[i],' ');
       msgln('');
     end; *)
     Er := 3;
     if __BlockModeOn then begin
       while TryCommand do
          Er := __SetStepPeriods(StepPeriodsArray);
       inc(QueueCount,count*2+2);
     end else begin
       while TryCommand do
          Er :=
__AddChangeTimebase(StepPeriodsArray.periods[0]);
       inc(QueueCount,3);
     end;
     SIHerror;
  end;
  PerformRedraw;
end;

procedure ScanDelay;
begin
  _Scan_Delay := WordArgument;
end;

procedure JumpSize;
begin
  _Jump_Size := WordArgument;
  if _Jump_Size < 50 then _Jump_Size := 50;    { limit }
end;

procedure JumpDelay;
begin
```

```
  _Jump_Delay := WordArgument;
end;

procedure LaserOn;
begin
  _Laser_On_Delay := TimeArgument;
  Er := __SetTurnOnDelay(_Laser_On_Delay);
  SIHerror;
end;

procedure LaserOff;
begin
  _Laser_Off_Delay := TimeArgument;
  Er := __SetTurnOffDelay(_Laser_Off_Delay);
  SIHerror;
end;

procedure SequenceMode;
begin
  { no-operation }
end;

procedure AbsoluteMode;
begin
  { no-operation }
end;

procedure Execute;
begin
  PerformRedraw;
  Er := 3;
  while TryBlockCommand do
     Er := __Execute;
  SIHerror;
  ResetRedraw;
end;
```

```
procedure ExecuteClear;
begin
  PerformRedraw;
  Er := 3;
  while TryBlockCommand do
     Er := __ExecuteClear;
  QueueCount := 0;
  SIHerror;
  ResetRedraw;
end;

procedure ClearTable;
begin
  Er := __Clear;
  QueueCount := 0;
  if Er<>5 then SIHerror;  { not in block mode okay }
  ResetRedraw;
end;

procedure SetMode;
begin
  if BinaryFlag then begin
     case CharArgument of
       'A': Er := __SetAutoShutterMode(CharArgument);
       'B': Er := __SetBlockMode(CharArgument);
       'P': Er := __SetPosAckMode(CharArgument);
     end;
  end else begin
     if length(TextCommand)=5 then case
          TextCommand[3] of
       'A': Er := __SetAutoShutterMode(TextCommand[5]);
       'B': Er := __SetBlockMode(TextCommand[5]);
       'P': Er := __SetPosAckMode(TextCommand[5]);
     end;
  end;
  SIHerror;
end;
```

```
procedure SoftReset;
begin
  Er := __SetAutoShutterMode('-');  { off }
  SIHerror;
  Er := __SetBlockMode('-');        { off }
  SIHerror;
  Er := __SetPosAckMode('-');     { off }
  SIHerror;
  QueueCount := 0;
end;

procedure HardReset;
begin
  SoftReset;
  Er := __ResetCmdQueue;
  SIHerror;
end;

procedure RedrawCount;
begin
  _Redraw_Count := WordArgument;
  if _Redraw_Count < 1 then _Redraw_Count := 1
     else if _Redraw_Count > 10 then _Redraw_Count
          := 10;
  SetRedrawCount(_Redraw_Count);
  PerformRedraw;
  Er := 3;
  while TryCommand do
     Er := __SetRedrawCount(_Redraw_Count);
  inc(QueueCount,3);
  SIHerror;
end;

procedure RedrawDelay;
begin
  _Redraw_Delay := WordArgument;
  Er := 3;
```

```
  while TryCommand do
     Er := __SetRedrawDelay(_Redraw_Delay);
  inc(QueueCount,3);
  SIHerror;
end;

procedure RedrawSize;
begin
  _Redraw_Size := WordArgument;
end;

procedure RivetCount;
begin
  _Rivet_Count := WordArgument;
  if _Rivet_Count < 1 then _Rivet_Count := 1
     else if _Rivet_Count > 10 then _Rivet_Count := 10;
  _Rivet_Count := _Rivet_Count - 1;
end;

procedure RivetPeriods;
var
  i: Integer;
begin
  if BinaryFlag then begin
     count := WordArgument;
     RivetMainPeriod := TimeArgument;
     for i:=0 to count-1 do RivetPeriodsArray.periods[i]
:= TimeArgument;
  end else begin
     RivetMainPeriod := TimeArgument;
     count := 0;
     while (Operand <> '') and (count<10) do begin
       RivetPeriodsArray.periods[count] := TimeArgument;
       inc(count);
     end;
  end;
  RivetPeriodsArray.num := count;
```

```
(*  if DebugMode then begin
     msg('New rivet periods array: ');
     msg(' Main=',RivetMainPeriod);
     for i:=0 to count-1 do msg('
',RivetPeriodsArray.periods[i]);
     msgln('');
  end; *)
end;

procedure RivetReduction;
begin
  _Rivet_Reduction := WordArgument;
end;

procedure ResetFromRivets;
begin
  if RivetPeriodsArray.num>0 then begin
     { set standard step periods }
     Er := 3;
     while TryCommand do
       Er := __SetStepPeriods(StepPeriodsArray);
     inc(QueueCount,StepPeriodsArray.num*2+2);
     SIHerror;
  end;
end;

procedure RivetX;
begin
  newX(PositionArgument);
end;

procedure RivetY;
var
  StartX,StartY,EndX,EndY: Real;
  ReducedStartX,ReducedStartY,ReducedEndX,ReducedEndY:
Real;
```

```
  ReductionX,ReductionY: Real;
  DirX,DirY,ReducedDirX,ReducedDirY: Integer;
begin
  StartX := oldX;
  StartY := Ypos;
  EndX   := Xpos;
  EndY   := PositionArgument;
  { set rivet step periods }
  Er := 3;
  while TryCommand do
     Er := __AddChangeTimeBase(RivetMainPeriod);
  inc(QueueCount,3);
  SIHerror;
  { draw complete vector }
  NextY;
  DirX := sgn(EndX - StartX);
  DirY := sgn(EndY - StartY);
  ReductionX := _Rivet_Reduction * deltaX;
  ReductionY := _Rivet_Reduction * deltaY;
  ReducedStartX := StartX + ReductionX;
  ReducedStartY := StartY + ReductionY;
  ReducedEndX   := EndX   - ReductionX;
  ReducedEndY   := EndY   - ReductionY;
  ReducedDirX := sgn(ReducedEndX - ReducedStartX);
  ReducedDirY := sgn(ReducedEndY - ReducedStartY);
  DoingRivet := true;
  { do rivet if reduced vector size is greater than 0
units }
  if ((ReducedDirX = DirX) and (ReducedDirY = DirY))
     and (_Rivet_Count > 0) then begin
     { end standard redraw if any }
     PerformRedraw;
     if RivetPeriodsArray.num>0 then begin
       { set rivet step periods }
       Er := 3;
       while TryCommand do
          Er := __SetStepPeriods(RivetPeriodsArray);
```

```
          inc(QueueCount,RivetPeriodsArray.num*2+2);
          SIHerror;
        end;
        { set rivet pass count }
        SetRedrawCount(_Rivet_Count);
        { draw reduced vector - the rivet }
        PositionJump(ReducedStartX,ReducedStartY);
        PositionMove(ReducedEndX,ReducedEndY);
        { end rivet redraw }
        PerformRedraw;
        { restore original redraw pass count }
        SetRedrawCount(_Redraw_Count);
        { jump to end of full vector }
        PositionJump(EndX,EndY);
    end;
    { reset standard step periods }
    Er := 3;
    while TryCommand do
       Er := __SetStepPeriods(StepPeriodsArray);
    inc(QueueCount,StepPeriodsArray.num*2+2);
    SIHerror;
    DoingRivet := false;
end;

procedure RivetPosition;
begin
    RivetX;
    RivetY;
end;

procedure WaitIdle;
var
    time: Integer;
begin
    time := WordArgument;
    if time < 100 then time := 100;  { set minimum delay
time }
```

```
  delay(time);  { delay long enough for background task
to see EX }
  repeat
  until __GetBkgndIdleStatus or keypressed;
  if keypressed then if StatusKey = CtrlBreak then
KeyAbort := true;
end;

{
   Drift Correction Commands
}

procedure SetDrift;
begin
  if BinaryFlag then DriftCorrectionFlag :=
(CharArgument = '+')
     else DriftCorrectionFlag := (TextCommand[3] = '+');
end;

procedure Reference1;
begin
  Sensor[1].Ref.X := RealArgument;
  Sensor[1].Ref.Y := RealArgument;
end;

procedure Reference2;
begin
  Sensor[2].Ref.X := RealArgument;
  Sensor[2].Ref.Y := RealArgument;
end;

procedure DriftAdj1;
begin
  Sensor[1].Adj.X := RealArgument;
  Sensor[1].Adj.Y := RealArgument;
  CorrectForDrift;
end;
```

```
procedure DriftAdj2;
begin
  Sensor[2].Adj.X := RealArgument;
  Sensor[2].Adj.Y := RealArgument;
  CorrectForDrift;
end;


{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


     STEREO Command Parsers

{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure CallOperation(OpNum: Integer);
begin
  case OpNum of
      1: BeginX;              { BX }
      2: BeginY;              { BY }
      3: JumpX;               { JX }
      4: JumpY;               { JY }
      5: NextX;               { NX }
      6: NextY;               { NY }
      7: StepSize;            { SS }
      8: StepPeriods;         { SP }
      9: ScanDelay;           { SD }
     10: JumpSize;            { JS }
     11: JumpDelay;           { JD }
     12: LaserOn;             { LO }
     13: LaserOff;            { LF }
     14: SequenceMode;        { NS }
     15: AbsoluteMode;        { AB }
     16: Execute;             { EX }
     17: ExecuteClear;        { EC }
     18: ClearTable;          { CL }
     19: SetMode;             { MD }
```

```
        20: SoftReset;              { SR }
        21: HardReset;              { HR }
        22: RedrawCount;            { RC }
        23: RedrawDelay;            { RD }
        24: RedrawSize;             { RS }
        25: RivetCount;             { VC }
        26: RivetPeriods;           { VP }
        27: RivetReduction;         { VR }
        28: RivetX;                 { VX }
        29: RivetY;                 { VY }
        30: WaitIdle;               { WI }
        31: SetDrift;               { DC }
        32: Reference1;             { R1 }
        33: Reference2;             { R2 }
        34: DriftAdj1;              { D1 }
        35: DriftAdj2;              { D2 }
        36: BeginPosition;          { BG }
        37: JumpPosition;           { JP }
        38: MoveToPosition;         { MV }
        39: RivetPosition;          { RV }
      else Return := 128;           { unrecognized command }
  end;
end;

procedure BinaryParse;
var
  OpNum: Integer;
begin
  { break apart multiple commands }
  done := false;
  while (not KeyAbort) and (not done) and (Return = 0)
do begin
    OpNum := Mem[CommandSeg:CommandOfs];
    if OpNum <> 0 then begin    { cmd of 0 marks end }
      inc(CommandOfs);
      CallOperation(OpNum);
    end else done := true;
```

```
  end;
end;

procedure ParseTextCommand;
var
  OpName: str4;
begin
  OpName := copy(TextCommand,1,2);
  Operand := copy(TextCommand,3,99);
  i := pos(OpName,OperationMnemonics);
  CallOperation((i + 2) div 4);    { good cmds give
1,2,3... }
end;

procedure TextParse;
begin
  { break apart multiple command strings }
  TextCommand := '';
  done := false;
  while (not KeyAbort) and (not done) and (Return = 0)
do begin
     Ch := chr(Mem[CommandSeg:CommandOfs]);
     done := Ch < ' ';
     if done or (Ch=';') then begin
       inc(CommandOfs);
       ParseTextCommand;
       TextCommand := '';
     end else begin
       if Ch in ['!'..'~'] then TextCommand :=
TextCommand + upcase(Ch);
       inc(CommandOfs);
     end;
  end;
end;
```

```
{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

     Process STEREO Commands coming through the
          Software Interrupt

     - Software Interrupt $65 (101) Handler -

     ES:BX points to first character of string.  String
should end with a character value of 0, 10 (LF), or 13
(CR).  An error code will be returned in AL with 0
meaning no error occured.

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure
ReceiveCommand(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP:
Word);
interrupt;
begin
  KeyAbort := false;
  CommandOfs := BX;
  CommandSeg := ES;
  INLINE ($FB);   { STI  ;allow interrupts }

  Return := 0;
  QueueCount := 0;
  case AX of
     0: begin
           BinaryFlag := false;
           TextParse;
        end;
     1: begin
           BinaryFlag := true;
           BinaryParse;
        end;
     else Return := 129;   { invalid STEREO request }
  end;
```

```
  CX := QueueCount;
  AX := Return;
  BX := CommandOfs;
end;

{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    Memory Resident Routines
    After the following routines install the
    memory-resident routines above, they are thrown
    away and not used again.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure ResidentTermination;
begin
  with regs do begin
    AH := $31;
    AL := 1;
    DX := trunc(StereoSize) shr 4;
    MsDos(regs);
  end;
end;

procedure InitMemoryResidency;
begin
  SetIntVec(_STEREOVECTOR,@ReceiveCommand);
end;

{   -  -  -  -  -   }

function WhiteSpace(var str: String;loc: Integer):
Integer;
begin
  while (loc <= length(str)) and (str[loc] in [#9,' '])
do loc := loc + 1;
  WhiteSpace := loc;
end;
```

```
procedure CheckForOptions(var str: String);
var
  i: Integer;
begin
  str := upper(str);
  while (length(str) > 1) and (str[1] = ' ') do
delete(str,1,1);
  i := pos('/D',str);   { set debug mode }
  if i>0 then begin
     DebugMode := true;
     delete(str,i,2);
  end;
  i := pos('/NV',str);  { not constant vector velocity }
  if i>0 then begin  { larger of X and Y axes }
     VelocityFlag := false;
     delete(str,i,3);
  end;
  i := pos('/NG',str);  { no geometric correction }
  if i>0 then begin
     GeoCorrectFlag := false;
     delete(str,i,3);
  end;
  i := pos('/NC',str);  { no version check }
  if i>0 then begin
     VersionCheckFlag := false;
     delete(str,i,3);
  end;
  i := pos('/SWAPXY',str);  { swap X & Y coordinates }
  if i>0 then begin
     SwapXY := true;
     delete(str,i,7);
  end;
  i := pos('/AXESBEGIN',str);  { axes-control begin
(home) coordinates }
  if i>0 then begin
     SwapBegin := true;
     delete(str,i,10);
```

```
    end;
    i := pos('/GEO',str);  { geo-correct axes order }
    if i>0 then begin
       j := WhiteSpace(str,i+4);
       case str[j] of
         'X': FirstX := true;
         'Y': FirstX := false;
         'N': GeoCorrectFlag := false;
       end;
       delete(str,i,j+2-i);
    end;
    i := pos('/X',str);  { X-axis orientation }
    if i>0 then begin
       NegX := (str[i+2]='-');
       delete(str,i,3);
    end;
    i := pos('/Y',str);  { Y-axis orientation }
    if i>0 then begin
       NegY := (str[i+2]='-');
       delete(str,i,3);
    end;
end;

procedure ShowAxis(Neg: Boolean;AxCh: Char);
begin
  if Neg then write('-') else write('+');
  write(AxCh,' axis');
end;

procedure ShowGeometry;
begin
  write('Geometry is ');
  if SwapXY then write('XY Swapped, ');
  if SwapBegin then write('Begin Swapped, ');
  if FirstX then begin
     ShowAxis(NegX,'X');
     write(' first, then ');
```

```
      ShowAxis(NegY,'Y');
    end else begin
      ShowAxis(NegY,'Y');
      write(' first, then ');
      ShowAxis(NegX,'X');
    end;
    writeln('.');
    if GeoCorrectFlag then writeln('Geometric Correction
active.');
end;

{  - - -  M A I N   P R O G R A M  - - -  }

begin
    CheckBreak := false;
    FillChar(FirstVar,ofs(LastVar)-ofs(FirstVar),#0);
       { clear variables }
    VelocityFlag := true;    { default to constant vector
        velocity }
    GeoCorrectFlag := true;    { default to geometric
        correction }
    VersionCheckFlag  := true;
    writeln;
    writeln(#213,many(#205,45),#184);
    writeln(#179,'  STEREO, Stereolithography Device
            Driver    ',#179);
    writeln(#179,'  Version ',ThisStereoVersion,'
            ',#179);
    writeln(#179,'  Copyright (C) 1988 by Tarnz
             Technologies   ',#179);
    writeln(#179,'  3D Systems Laser Stereolithography
             System  ',#179);
    writeln(#212,many(#205,45),#190);
    writeln;
    cmd := StartOfProgramCode;
    ReadCommandLine(cmd);
    if not __LaserPresent then goto ExitProgram;
```

```
  if __StereoOk then begin
     writeln('STEREO Device Driver Version
',StereoVersion,
       ' already installed.');
     goto ExitProgram;
  end;
  CheckForOptions(cmd);
  if VersionCheckFlag then
     if LaserVersion < MinSystemVersion then begin
       writeln('Obsolete LASER system in use.');
       goto ExitProgram;
     end;
  Xpos       := 32767;
  Ypos       := 32767;
  DoingRivet := false;
  RivetMainPeriod      := 1190;
  StepPeriodsArray.num  := 0;
  RivetPeriodsArray.num := 0;
  OperationMnemonics := '*BX *BY *JX *JY *NX *NY *SS *SP
*SD *JS *JD *LQ '
     + '*LF *NS *AB *EX *EC *CL *MD *SR *HR *RC *RD *RS
*VC *VP *VR *VX '
     + '*VY *WI *DC *R1 *R2 *D1 *D2 *BG *JP *MV *RV';
  Sensor[1].Ref.X := 0;
  Sensor[1].Ref.Y := 0;
  Sensor[2].Ref.X := 65535;
  Sensor[2].Ref.Y := 65535;

  { read and/or set defaults }
  Assign(DefaultFile,'STEREO.DEF');
  {$I-}
  reset(DefaultFile);
  {$I+}
  if IOresult<>0 then begin
     writeln('Can''t find STEREO.DEF default file!');
     _Step_Size        := 32;
     _Step_Period    := 119;
```

```
        _Scan_Delay      := 64;
        _Jump_Size        := 65535.0;
        _Jump_Delay      := 100;
        _Laser_On_Delay  := 0;
        _Laser_Off_Delay := 0;
        _Redraw_Count  := 1;
        _Redraw_Delay  := 0;
        _Redraw_Size    := 200;
        _Rivet_Count    := 1;
        _Rivet_Reduction := 100;
      end else begin  { read default file }
        i := 1;
        while (i<=14) and not eof(DefaultFile) do begin
          readln(DefaultFile,temp);
          if not ((length(temp)=0) or (temp[1]='!')) then
begin
            case i of
              1: IntParse(_Step_Size);
              2: begin
                   count := 0;
                   while (length(temp)>0) and (count<10)
                     do begin
                     IntParse(StepPeriodsArray.periods
                     [count]); count := count+1;
                   end;
                   StepPeriodsArray.num := count;
                   _Step_Period := StepPeriodsArray
                     .periods[0];
                 end;
              3: IntParse(_Scan_Delay);
              4: RealParse(_Jump_Size);
              5: IntParse(_Jump_Delay);
              6: IntParse(_Laser_On_Delay);
              7: IntParse(_Laser_Off_Delay);
              8: IntParse(_Redraw_Count);
              9: IntParse(_Redraw_Delay);
             10: IntParse(_Redraw_Size);
```

```
            11: IntParse(_Rivet_Count);
            12: begin
                  count := 0;
                  while (length(temp)>0) and (count<10)
do begin

IntParse(RivetPeriodsArray.periods[count]);
                     count := count+1;
                  end;
                  RivetPeriodsArray.num := count;
                end;
            13: RealParse(_Rivet_Reduction);
          end;
          i := i+1;
       end; {if not}
     end; {while i<}
     close(DefaultFile);
  end;
  CurrentRedrawCount := _Redraw_Count;

  XGeoCorrectTable := HeapPtr;
  if GeoCorrectFlag then begin
   { read Geometric Correction Table }
     Assign(GeoFile,'STEREO.GEO');
     {$I-}
     reset(GeoFile);
     {$I+}
     if IOresult<>0 then begin
       writeln('Can''t find STEREO.GEO Geometric
Correction File!');
       GeoCorrectFlag := false;
     end else begin  { read geometric correction table
from file }
       repeat
          readln(GeoFile,temp);
          if upper(copy(temp,1,7)) = '!SENSOR' then
begin
```

```
                val(temp[8],SensorNum,code);
                temp := copy(temp,9,99);
                if SensorNum in [1..2] then begin
                   RealParse(Sensor[SensorNum].Ref.X);
                   RealParse(Sensor[SensorNum].Ref.Y);
                end;
                temp := '!';
             end;
          until ((length(temp)>0) and (temp[1]<>'!')) or
eof(GeoFile);
          SkipToNum(temp);
          IntParse(XCount);
          SkipToNum(temp);
          IntParse(YCount);
          YGeoCorrectTable :=
PtrAdd(XGeoCorrectTable,LongInt(XCount*YCount) shl 2);
          SkipToNum(temp);
          IntParse(CoordCount);
          SkipToNum(temp);
          IntParse(TableDivisor);
          If TableDivisor < 1 then TableDivisor := 1024;  {
for V2.30 compatibility }
          if (XCount<>YCount) or (CoordCount<>2)
          or (XCount>MaxGeo) or (YCount>MaxGeo) then begin
             writeln('Illegal format for STEREO.GEO
file.');
             halt(1);
          end;
          XC := 0;
          YC := 0;
          StoreCount := 0;
          while (XC<XCount) and not eof(GeoFile) do begin
             while (YC<YCount) and not eof(GeoFile) do
begin
               repeat
                  readln(GeoFile,temp);
```

```
            until ((length(temp)>0) and (temp[1]<>'!'))
or eof(GeoFile);
            SkipToNum(temp);
            RealParse(Xcoord);
            SkipToNum(temp);
            RealParse(Ycoord);
            XGeoCorrectTable^[StoreCount] :=
trunc(Xcoord*GeoMult);
            YGeoCorrectTable^[StoreCount] :=
trunc(Ycoord*GeoMult);
            inc(StoreCount);
            inc(YC);
          end; {while YC}
          inc(XC);
          YC := 0;
        end; {while XC}
        if XC < XCount then begin
          writeln('Not enough entries in STEREO.GEO
file.');
          halt(2);
        end;
        while not eof(GeoFile) do begin
          readln(GeoFile,temp);
          if (length(temp)>0) and (temp[1]<>'!') then
begin
            SkipToNum(temp);
            if temp[1] in ['0'..'9','.'] then begin
              writeln('Too many entries in STEREO.GEO
file.');
              halt(3);
            end;
          end;
        end;
        close(GeoFile);
      end;
    end;
    for i:=1 to 2 do Sensor[i].Adj := Sensor[i].Ref;
```

```
  CorrectForDrift;   { gain is 1, offset is 0 }
  StereoSize := MemAbs(XGeoCorrectTable) +
(LongInt(XCount*YCount) shl 3)
     - (LongInt(PrefixSeg) shl 4) + 1024;
  StereoSize := ((StereoSize shr 4)+1) shl 4;
(*
  writeln('MemAbs XGeo=',MemAbs(XGeoCorrectTable));
  writeln('MemAbs YGeo=',MemAbs(YGeoCorrectTable));
  writeln('X*Y shl 2=',LongInt(XCount*YCount) shl 2);
  writeln('PrefixSeg=',LongInt(PrefixSeg));
  writeln('StereoSize=',StereoSize);
*)
  writeln('STEREO Device Driver installed using ',
     RealStr(StereoSize/1024),'K bytes.');
  ShowGeometry;

  InitMemoryResidency;
  ResidentTermination;

ExitProgram:
end.
```

```
STERCMDS.INC
{
STEREO V2.80 Binary Command Equates
}
const
  STEREO_BX = 1;
  STEREO_BY = 2;
  STEREO_JX = 3;
  STEREO_JY = 4;
  STEREO_NX = 5;
  STEREO_NY = 6;
  STEREO_SS = 7;
  STEREO_SP = 8;
  STEREO_SD = 9;
  STEREO_JS = 10;
  STEREO_JD = 11;
  STEREO_LO = 12;
  STEREO_LF = 13;
  STEREO_NS = 14;
  STEREO_AB = 15;
  STEREO_EX = 16;
  STEREO_EC = 17;
  STEREO_CL = 18;
  STEREO_MD = 19;
  STEREO_SR = 20;
  STEREO_HR = 21;
  STEREO_RC = 22;
  STEREO_RD = 23;
  STEREO_RS = 24;
  STEREO_VC = 25;
  STEREO_VP = 26;
  STEREO_VR = 27;
  STEREO_VX = 28;
  STEREO_VY = 29;
  STEREO_WI = 30;
  STEREO_DC = 31;
  STEREO_R1 = 32;
```

```
STEREO_R2 = 33;
STEREO_D1 = 34;
STEREO_D2 = 35;
STEREO_BG = 36;
STEREO_JP = 37;
STEREO_MV = 38;
STEREO_RV = 39;
```

```
'=======================================================
'  3D Systems Stereolithography System Software
'=======================================================
'
'  TESTPART.BAS
'  SLA-1 Test Part Utility Software
'
'-------------------------------------------------------
'  Compile with TBM - Modified Turbo Basic - not TB!
'-------------------------------------------------------
'
'  Recent History:
'   1/11/88  Ver 2.50  first release
'   1/25/88  Ver 2.60  version update only
'   3/09/88  Ver 2.80  added copyright notice
'   4/11/88  Ver 3.00  production release
'
'-------------------------------------------------------

  VersionId$ = "3.00"

  SingleBar$ = STRING$(45,196)
  CLS
  PRINT SingleBar$
  PRINT "  TESTPART, Test Part Utility"
  PRINT "  Version "+VersionId$
  PRINT "  Copyright (C) 1988 by 3D Systems, Inc."
  PRINT "  3D Systems Laser Stereolithography System"
  PRINT SingleBar$
  PRINT
  PRINT
  PRINT "   1. BanjoTop, Determine Working Curves"
  PRINT
  PRINT "   X. Exit Program"
  PRINT
  PRINT
KeyLoop:
```

```
  K$ = UCASE$(INKEY$)
  IF K$ = "1" THEN StartBanjo
  IF K$ = "X" THEN END
  GOTO KeyLoop

'
'*********************************************************
'  BANJOTOP       turbobasic version      11/19/87     wav
'
'*********************************************************

StartBanjo:
      CLS
      PRINT
      LOCATE 1,36
      PRINT "BANJOTOP"
      PRINT
     INPUT "INPUT BIGGEST SP VALUE OF 5 TO BE DRAWN
     (160 minimum) :",sp$
            t$ = ucase$(left$(sp$,1))
            if t$="Q" or t$="X" then end
            sp = VAL(sp$)
            IF sp/16 < 10 THEN
              PRINT
              PRINT
              PRINT " ERROR  ERROR  ERROR  ERROR"
              PRINT " THE SMALLEST SP MUST NOT BE LESS
THAN 10"
              PRINT
              PRINT "THUS, USE SP 160 FOR FASTEST LASER
                DRAWING !"
              LOCATE 20,30
              PRINT "ANY KEY TO TRY AGAIN ...";
              GOSUB strobe
              GOTO StartBanjo
            END IF
          PRINT "THE FOLLOWING STRING SP'S WILL BE
```

```
                GENERATED."
               PRINT
                 FOR I = 1 TO 5
                      PRINT "SP ";sp
                      sp = sp/2
                 NEXT I
               PRINT
               PRINT "OK ? (Y/N)"
               gosub strobe
               IF ans$ <>"Y" AND ans$<> "y" THEN StartBanjo

               CLS
               SP = 32*SP
               STRINGSP = SP

               PRINT "PLEASE WAIT..."
               SHELL "CLIFF 'HR/W/T"

mainprogram:
                    GOSUB setup
                    GOSUB mainmenu
                    GOSUB openfile
                    GOSUB writeparameters
                    GOSUB writebase
                    GOSUB writestrings
                    LOCATE 18,27
                    PRINT "making base and strings...";
                    GOSUB movemirrors
                    LOCATE 22,35
                    PRINT " done !!":
                    END

END
writebase:
                M$ = "J"
                X = xside/2
                Y = yside/2 + yinc
```

```
                GOSUB writexytofile
                M$ = "N"
                X =  xside/2
                Y = -yside/2
                GOSUB writexytofile
                X = -xside/2
                Y = -yside/2
                GOSUB writexytofile
                X = -xside/2
                Y =  yside/2
                GOSUB writexytofile
                X =  xside/2 + yinc
                Y =  yside/2
                GOSUB writexytofile
                RETURN

writestrings:

            sp = stringsp * 2
              FOR x = -3*xside/8 TO xside/8 STEP
6*xside/(12*4)
                sp = sp/2
                PRINT#2,"SP";sp
                M$="J"
                Y= -yside/2 - yinc
                GOSUB writexytofile
                M$="N"
                Y=  yside/2 + yinc
                GOSUB writexytofile
              NEXT X
            RETURN

writexytofile:
              PRINT#2, M$;"X";cx+x
              PRINT#2, M$;"Y";cy+y
              RETURN
writeparameters:
```

```
                    PRINT#2,"MD BL +"
                    PRINT#2,"CL"
                    PRINT#2,"SS2"
                    PRINT#2,"SP";sp
                    PRINT#2,"JD0"
                    PRINT#2, "BX32000"
                    PRINT#2, "BY62000"
                    RETURN

movemirrors:

                    PRINT#2,"EX"
                    CLOSE
                    SHELL "CLIFF LASER.DAT/W/T"
                    ON ERROR GOTO chkerror
                    OPEN "CLIFF.ERR" FOR INPUT AS #3
                    CLOSE #3
                    PRINT "CLIFF process aborted."
                    END

END

chkerror:
                    CLOSE #3
                    RESUME chkdone

chkdone:
                    ON ERROR GOTO 0
                    RETURN


openfile:

                OPEN "LASER.DAT" FOR OUTPUT AS #2
                RETURN
mainmenu:
```

```
                CLS
                  FOR I=1 TO 79
                    LOCATE 1,I,0
                    PRINT "*";
                  NEXT
                LOCATE 2,35,0
                PRINT "*   SLA1   *"
                LOCATE 3,33,0
                PRINT "PROGRAM: BANJOTOP"
                  FOR I=1 TO 79
                    LOCATE 4,I,0
                    PRINT "*";
                  NEXT
                  FOR I = 1 TO 5
                    LOCATE 10 + i,30
                    PRINT "SP ";STRINGSP
                    STRINGSP = STRINGSP / 2
                  NEXT I
                STRINGSP = 32 * STRINGSP
                LOCATE 23,25
                PRINT "ANY KEY TO CONTINUE...";
                GOSUB strobe
                  FOR I=25 TO 55
                    LOCATE 23,I,0
                    PRINT " ";
                  NEXT
                RETURN

setup:

                cx = 32000              ' approx center of pot
                cy = 32000
                scale = 140             ' bits/mm
                xside = 25              ' 1 x 1/2 inch
                xside = xside * scale
                yside = xside / 2
                sp = 500                '  base step period
```

```
            yinc = yside / 4      '  overlap
            RETURN

strobe:
        ans$ = INKEY$
        IF LEN(ans$) = 0 then strobe
        return
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================
   XFER.PAS
   SLA-1 File Transfer Utility Software

---------------------------------------------------------
   History:
   11/12/87  Ver 2.30  first Beta software release
   12/02/87  Ver 2.31  reactivates tcp for Turbo Pascal
                       V4.0 Menu
   12/17/87  Ver 2.40  telnet/ftp host is 'Slice
                       Computer' (not 'process')
    1/11/88  Ver 2.50  support of different Ethernet
                       types based on SLA1.SYS file
    1/20/88  Ver 2.51  upper case X for exit
    1/25/88  Ver 2.60  version update only
    3/09/88  Ver 2.80  added copyright notice
    4/04/88  Ver 3.00  change menus

---------------------------------------------------------}
{$R-,S+,I+,D+,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,16384 }

uses Crt,Dos;

type
  EthernetSupport =
(_NoEthernet,_Excelan,_Micom,_Unknown);

var
  Ch: Char;
  Line,SingleBar: String;
  Ethernet: EthernetSupport;
  SystemFile,ErrorFile: Text;

const
```

```
  VersionId = '2.80';
  ComSpec = 'C:\COMMAND.COM';
  SystemFileName = 'C:\SLA1.SYS';
  ErrorFileName = 'C:\SYSTEM.ERR';

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln(Text);
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'XFER: ',Text);
  close(ErrorFile);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): String;
var
  temp: String;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;
```

```
procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

procedure DetermineEthernetSupport;
begin
  Ethernet := _NoEthernet;
  assign(SystemFile,SystemFileName);
  {$I-}
  reset(SystemFile);
  {$I+}
  if IOresult=0 then begin
    while not eof(SystemFile) do begin
      readln(SystemFile,Line);
      Line := upper(Line);
      if copy(Line,1,9)='ETHERNET=' then begin
        case Line[10] of
          'N': Ethernet := _NoEthernet;
          'E': Ethernet := _Excelan;
          'M': Ethernet := _Micom;
          'U': Ethernet := _Unknown;
        end;
      end;
    end;
  end;
end;

begin
  DetermineEthernetSupport;
  SingleBar := many(#196,26);
  if Ethernet = _Excelan then Exec(ComSpec,'/C REX');
  repeat
    ClrScr;
    centerln('XFER, Ethernet File Transfer');
    centerln('Version '+VersionId);
```

```
    centerln('Copyright (C) 1988 by 3D Systems, Inc.');
    centerln('3D Systems Laser Stereolithography
System');
    centerln(SingleBar);
    writeln;
    writeln;
    writeln;
    if Ethernet = _NoEthernet then begin
      SystemError('No Ethernet Communications Support');
      halt(1);
    end else begin
      case Ethernet of
        _Excelan: centerln('using Excelan Ethernet
         support');
        _Micom  : centerln('using Micom Interlan
         support');
        else centerln('using unknown support');
      end;
      writeln;
      writeln;
      centerln('1. TELNET, Terminal Utility     ');
      centerln('2. FTP,    File Transfer Program');
      centerln('X. Exit Program                 ');
      writeln;
      writeln;
      gotoxy(32,whereY);
      write('Which action? ');
      Ch := upcase(ReadKey);
      if Ch in ['1','2','X'] then begin
        writeln;
        writeln;
        case upcase(Ch) of
          '1': begin
                 Exec(ComSpec,'/C TELNET
SliceComputer');
                 halt(0);
               end;
```

```
                '2': begin
                       Exec(ComSpec,'/C FTP SliceComputer');
                       halt(0);
                     end;
                'X': halt(0);
              end;
            end;
          end;
        until 0=1;
      end.
```

```
{=========================================================
     3D Systems Stereolithography System Software
=========================================================}
     ZSTAGE.PAS
     SLA-1 Z-STAGE CONTROL PROGRAM
---------------------------------------------------------
     Recent History:
     11/12/87  Ver 2.30  first Beta software release
     11/15/87  Ver 2.31  fixed motor controller timing
                         eliminated need for NUM LOCK
      1/22/88  Ver 2.50  added motion status
                         can move specified distances
                         (inches) supports ZSTAGE.PRM
                         file
      1/25/88  Ver 2.60  version update only
      2/24/88  Ver 2.61  Home Directory search for
                         ZSTAGE.PRM FIX displays source
                         of ZSTAGE.PRM file
      3/22/88  Ver 3.00  converted from Turbo Basic to
                         Turbo Pascal Version 4.0
      4/14/88  Ver 3.01  let any key stop the elevator
                         (dumb key)
      4/15/88  Ver 3.02  fix acceleration and velocity
                         send commands

---------------------------------------------------------}
uses CRT,DOS;

const
     VersionId      = '3.02';
     ParamFileName  = 'ZSTAGE.PRM';
     HomeDirectory  = 'c:\3DSYS\';
     Indent         =  19; {use 22 for top lines of
                            the display }
     Ctr1C          = ^C;
     Cr             = ^M;
     afterDecimal   =  4;
```

```
type
     str8  = string[ 8];
     str16 = string[16];
     str25 = string[25];
     str80 = string[80];

var
     ParamFileLine : String;
     CMD : str80;
     OpenParamFile, IndentStr : str25;
     InchDistanceStr, Accel, Veloc : str8;
     TimeoutValue, ZAddress, i, m, n, ValCode, IOerr :
      Integer;
     InchDistance, r : real;
     ParamFile : Text;
     BigDist,MicroSteps, j, MinimumZ : LongInt;
     ShowStuff, NegZStage, ShowParms, TimeOut, dumbkey :
      Boolean;
     regs: registers;
     key, Dir : char;

{$I combine.inc}
{$R-,S-,I-,F-,V-,B-}

function RealVal(line: string;var code: Integer):Real;
var
     code2 : integer;
     num   : real;
begin
     if line[1]='.' then line := '0' + line;
     if (line[1]='-') and (line[2]='.') then begin
          line := '-0' + line;
          for i: = 4 to length(line) do line
                    [i-1]:=line[i];
          line[length(line)] := ' ';
     end;
     val(line,num,code);
```

```
        if code <> 0 then val(copy(line,1,code-1),
         num,code2);
        RealVal := num;
    end;

    procedure DefaultParms;
    begin
        for i:=1 to 255 do ParamFileLine[i]:=' ';
        NegZStage    := false;
        ShowStuff    := false;
        ShowParms    := false;
        ZAddress     := 800;
        Accel        := '0.5';
        Veloc        := '  1';
        BigDist      := 2000000;
        MicroSteps   := 100000;
        TimeoutValue := 2000;
    end;

    procedure ReadParms;
    begin
        OpenParamFile   := ParamFileName;
        assign(ParamFile,OpenParamFile);
        reset(ParamFile);
        IOerr := IOresult;
        if IOerr <> 0 then begin
            OpenParamFile := HomeDirectory+ParamFileName;
            assign(ParamFile,OpenParamFile);
            reset(ParamFile);
            IOerr := IOresult;
            if IOerr <> 0 then OpenParamFile := ' '
        end;
        if IOerr = 0 then begin
            i := 0;
            while not eof(ParamFile) do begin
                readln(ParamFile,ParamFileLine);
                ParamFileLine := upper(ParamFileLine);
```

```
                    if ParamFileLine[1] = '"' then begin
                         ParamFileLine[1] := ' ';
                         m := pos('"',ParamFileLine);
                         n := pos('/S',ParamFileLine);
                         if (0<n) and (n<m) then ShowStuff
                           := true;
                         n := pos('/NEGZ',ParamFileLine);
                         if (0<n) and (n<m) then NegZStage
                           := true;
                    end;
                    if (ParamFileLine[1] <> ' ') and
                         (ParamFileLine[1] <> '!') and
                         (ParamFileLine[1] <> '"') then begin
                         if i < 6 then begin
                              i := i+1; { this allows blank
                                   lines in the parameter
                                   file }
                              val(ParamFileLine,j,valcode);
                              case i of
                    1 : if valcode=0 then  ZAddress := j;
                    2 : Accel := ParamFileLine;
                    3 : Veloc := ParamFileLine;
                    4 : if valcode=0 then BigDist := j;
                    5 : if valcode=0 then MicroSteps := j;
                    6 : if valcode=0 then TimeoutValue := j;
                         end;
                    end;
               end;
          end;
     end;
end;
end; { ReadParams }

procedure ShowZStageMoverMenu;
     begin
          window(1,1,80,25);
          ClrScr;
```

```
            writeln(IndentStr,'                Elevator
    Controller                ');
            writeln(IndentStr,' 3D Systems
    Stereolithgraphy System');
        writeln(IndentStr,' Version '+VersionID+' 3D Systems,
    Inc.');
        gotoxy(1,5);
        if TIMEOUT then writeln(IndentStr,'   ELEVATOR OFF ')
        else                        writeln(IndentStr'
                                    'ELEVATOR STOP');
        DrawBox(Indent-1,7,80-Indent,21,1);
        window(Indent,9,79-Indent,21);
        writeln('      UP ARROW  move elevator up');
        writeln;
        writeln(' DOWN ARROW     move elevator down');
        writeln;
        writeln('  SPACE BAR     stop elevator motion');
        writeln;
        writeln('             D   move a specified distance');
        writeln;
        writeln;('        T toggle parameter display');
        writeln;
        writeln('           Q or X    exit to Utility menu');
        if ShowParms then begin
            window(Indent-6,22,80,25);
            if OpenParamFile = ' ' then
            writeln('      Not using any Parameter File:')
            else writeln('Using '+OpenParamFile+' Parameter
                         File:');
            write('        Address=',Zaddress,'
                Accel='+accel+' Veloc='+Veloc);
            if NegZStage then writeln('   /NEGZ') else
             writeln;
            writeln('Big Dist=',bigdist,'
                Microsteps=',microsteps,' Timeout Value);
        end;
```

```
end; { procedure ShowZStageMoverMenu }

procedure GetSelection;
begin
    key := 'Z';
    repeat
        HandleKeyboardRequest;
        key := upcase(key);
        if dumbkey then key:=' ';
        dumbkey:=false;
        if key in [#72,#80,' ','D','T','Q','X'] then exit; {
            up and down arrows }
    until false;
end; { procedure GetSelection }

procedure SendElevCommand(Ch: Char; TimeOutValue: integer;
    TimeOut: boolean);
var
    b, TC : integer;
begin
    TimeOut := false;
    TC              := 0;
    repeat    { Read status byte to determine if the input... }
        b := port[ZAddress+1]; { data buffer (IDB) is ready
for a character. }
        TC := TC + 1;
    until ((b and $10)>0) or (TC > TimeOutValue);      { Bit 4
        true=1 means ready. }
    port[ZAddress] := ord(Ch);     { Convert Character to
                                     ASCII, give to IDB }
    port[ZAddress+1] := $70; { Reset status byte for ready to
                               send to motor. }
    repeat
        b:= port[ZAddress+1];     {Read status byte to
                                   determine if the IDB... }
        TC := TC + 1;  { read the instruction, if not, wait
                         until ... }
```

```
    until ((b and $10)=0) or (TC > TimeOutValue); { it does. }
    port[ZAddress+1] := $60; { Reset status byte for ready to
                               send again. }
    if TC > TimeOutValue then TimeOut := True;

end; { Send Elevator Command }
procedure ElevatorInterface(Cmd: string;TimeOutValue:integer;
TimeOut:boolean); { this is the gateway through which all
elevator commands must pass }
var
    ii : integer;
begin
{ CMD := 'E MN A'+Accel+' V'+veloc+'+CMD; from SetAVD
procedure }
    if ShowStuff then begin
        writeln(' DEBUT Elevator Interface CMD='+Cmd);
        delay(1);
    end;
    delay(1); { this is required for the program configuration
              to work right }
    for ii := 1 to length(Cmd) do
        SendElevCommand(Cmd[ii],TimeOutValue,TimeOut);
end; { Elevator Interface }

procedure WaitforMoveComplete(Cmd:string;TimeOutValue:integer;
    TimeOut:boolean);
var
    b, response, TC : integer;
begin
    TC := 0;
    repeat                { Loop until we get desired response. }
        repeat
            b := port[Zaddress+1];      { Read status byte. }
        until ((b and $08)<>0) and (not timeOut);
        response := port[Zaddress];   { OK to read response
                                        from controller. }
```

```
            port[ZAddress+1] := $E0  { Set status byte to message
                                       received. }
            repeat
                 b := port[ZAddress+1];   { Wait for read status
                                            to hit ready. }
                 TC := TC + 1;
            until ((b and $08)=0) or (TC > TimeOutValue);
            port[ZAddress+1} := $60;        { Reset status byte. }
        until (chr(response) = CR) or (TC > timeoutValue);

        if ShowStuff then begin
            window(1,1,80,25);
            ClrScr;
            writeln('response=',response);
            writeln;
            writeln('Tc                    =',TC);
            delay(10000);
        end;
    end; { Wait for elevator controller response }

    procedure SetZTableAVD;
    begin
        CMD:='';
        CMD := 'E MN A'+Accel+' V'+veloc+' K ';
        ElevatorInterface(CMD,TimeOutValue,TimeOut);
    end;

    procedure MoveGivenDistance;
    var
        dir : str8;
    begin
        window(Indent-8,22,79,25);
        writeln('          *** key distance in inches (positive
             is up) *** ');
        writeln('                                                  ');
        writeln('                                                  ');
        gotoxy(Indent,2);
```

```
    readln(inchdistancestr);
    if inchdistancestr[1] <> ' ' then begin
        inchdistance := RealVal(inchdistancestr,valcode);
        if inchdistance <> 0 then begin
            if not NegZStage then inchdistance := -
inchdistance;
            j := trunc(inchdistance * microsteps);
            STR(j,inchdistancestr);
            CMD:='E MN A'+Accel+' V'+veloc+'
                D'+inchdistancestr+' G CR ';
            ElevatorInterface(CMD,TimeOutValue,TimeOut);
            dir := 'DOWN ';
            if ((j < 0) and (not NegZStage)) or ((j > 0) and
                NegZStage) then dir := 'UP ';
            window(1,5,80,6);
            if not TimeOut then writeln(IndentStr,' ELEVATOR
'+dir);
            WaitforMoveComplete(CMD,TimeOutValue,TimeOut);
        end;
    end;
end; { procedure MoveGivenDistance }

procedure MoveTable(direction:string);
begin
    MinimumZ := BigDist;
    if direction = 'UP ' then MinimumZ := -BigDist;
    CMD:='';
    CMD:='E MN A'+Accel+' V'+veloc+' D'+RealStr(MinimumZ,0)+
'G';
    window(1,5,80,6);
30  if not TimeOut then writeln(IndentStr+
'ELEVATOR'+direction);
    repeat
        ElevatorInterface(CMD,TimeOutValue,TimeOut);
        delay(45);
    until KeyPressed;
    dumbkey := true;
```

```
{ key:=Readkey; }
    ElevatorInterface('K ',TimeOutValue,TimeOut);
end; { procedure MoveTable }

begin { main program }
    ClrScr;
        window(1,1,80,25);
        IndentStr := many(' ',Indent);
        DefaultParms;
        ReadParms;
        CursorOff;
        dumbkey := false;
        TimeOut := false;
        InitZTable;
        SetZTableAVD;
        repeat
            ShowZStageMoverMenu;
            GetSelection;
            case key of
            #72 : if not NegZStage then MoveTable('UP  ')
else

MoveTable('DOWN');
            #80 : if not NegZStage then MoveTable('DOWN')
else

MoveTable('UP  ');
            'D' : MoveGivenDistance;
            'T' : ShowParms := not(ShowParms);
            'Q' : Abort;
            'X' : Abort;
        end;
    until false;
    ClrScr;

end. { main program }
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================
   COMBINE.INC
   SLA-1 This is the master include library to COMBINE
     shared procedures

   The following is a list and description of the
     subroutines:

   CursorOff - remove the cursor from the SLA display
     screen
   CursorOn  - show the cursor on the SLA display screen
   Beep      - produce a auditable beep

   Upper     - convert a string to all upper case
   Many      - propigate a charater a specified number
               of times in a string
   Drawbox   - draw a box of variable size which can be
               used to border text

   Abort     - abort the current operation and return to
               the previous task
   HandleKeyboardRequest - Read the key unless it is a
                cntl-c

   InitZTable - initialize the SLA platform
   ZTable     - access the port for the SLA platform
   ZMove      - send a command to the Z-axis stepper
                controller
   FormatZCommand - move the platform a specified
                    distance
   WaitForIdleZStage - keep the platform moving until
                       the space bar is pressed

---------------------------------------------------------}
```

```
procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
```

```
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function sgn(num: real): real;
begin
   if num<0 then sgn := -1
   else if num>0 then sgn := 1
   else sgn := 0;
end;

function RealStr(num: Real;AfterDecimal: Integer):
str80;
var
   i : integer;
   temp : str80;
begin
   Str(num:10:AfterDecimal,temp);
```

```
    i := 1;
    while (i<length(temp)) and (temp[i]=' ') do inc(i);
    RealStr := copy(temp,i,99);
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = Ctrl C then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;

procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                { Type        1    2    3    4     }
```

```
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;



{-------------------------------------------------------
                        Z-Stage Elevator Routines
-------------------------------------------------------}

procedure InitZTable;
```

```
begin
  port[ZAddress+1] := $64;
  port[ZAddress+1] := $60;
  delay(100);
  port[ZAddress+1] := $40;
  port[ZAddress+1] := $60;
  delay(100);
end;

 { procedure ZTable(Ch: Char);
var
  b: Integer;
begin
  repeat
    b := port[ZAddress+1];
  until (b and $10)>0;
  port[ZAddress] := ord(Ch);
  port[ZAddress+1] := $70;
  repeat
    B:=port[ZAddress+1];
  until (b and $10)=0;
  port[ZAddress+1] := $60;
end;

procedure ZMove(Cmd: string);
begin
  if ShowStuff then begin
    if whereX>1 then writeln;
    writeln('   ZTable>',Cmd);
  end;
  if not ZTableOff then begin
    Cmd := Cmd + Cr;
    for i:=1 to length(Cmd) do ZTable(Cmd[i]);
  end;
end;

procedure FormatZCommand(Distance: Real);
```

```
begin
  Distance := sgn(Distance) * int(abs(Distance) * ZPitch
* 1000 / 25.4);
  if NegZStage then Distance := -Distance;
  ZMove('D'+RealStr(Distance,0)+' G');
end;

procedure WaitForIdleZStage;
var
  b: Word;
begin
  repeat
    b := port[ZAddress+1];
  until keypressed or (b and 2>0);
end;   ******** }
```

APPENDIX K

© 1989 3D SYSTEMS, INC.

```
PFPCHL     H          1333     11-09-88      4:35p
PFPCURE    H           922     11-09-88      4:35p
PFPHI      H          3831     11-09-88      4:35p
PFPSEG     H          2282     11-09-88      4:35p
PFPTRI     H          1693     11-09-88      4:35p
PFPUTL     H           959     11-09-88      4:35p
STORAGE    H           897     11-09-88      4:35p
TPFP       C         40848     11-09-88      4:36p
TPFPCHL    C         19600     11-09-88      4:36p
TPFPCURE   C         25150     11-09-88      4:36p
TPFPIDB    C         27851     11-09-88      4:36p
TPFPTRI    C          8432     11-09-88      4:36p
TPFPTUTL   C         11161     11-09-88      4:36p
TPFPUTL    C          4679     11-09-88      4:36p
PFPGRAF    C          3098     11-09-88      4:37p
PFPHI      C         26335     11-09-88      4:37p
PFPIDB     C         27851     11-09-88      4:37p
PFPRIE     C          4318     11-09-88      4:38p
PFPRIF     C         16090     11-09-88      4:38p
PFPSEG     C          9175     11-09-88      4:38p
PFPZACT    C          3327     11-09-88      4:38p
PRTUTL     C         10091     11-09-88      4:38p
           25 File(s)       1290240 bytes free

C:\R310\SLICE>
           cc   tpfp.c tpfpchl.c pfpgraf.c pfphi.c
pfpidb.c tpfptri.c pfprie.c pfprif.c pfpseg.c tpfptutl.c
tpfputl.c pfpzact.c prtutl.c tpfpcure.c  -lm -o slice320
```

© 1989 3D SYSTEMS, INC.

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFP.H                                               */
/*                                                     */
/*******************************************************/

#include "../storage.h" /* Include storage definitions
*/

#define SYS_SUCCESS          0
#define SYS_FAILURE          1

#define DISPLAY_OFF          0
#define DISPLAY_ON           1

#define LB_CODE              1
#define LH_CODE              2
#define NFUB_CODE            3
#define NFUH_CODE            4
#define NFUF_CODE            5
#define NFDB_CODE            6
#define NFDH_CODE            7
#define NFDF_CODE            8
#define FDB_CODE             9
#define FDH_CODE             10
#define FDF_CODE             11
#define FUB_CODE             12
#define FUH_CODE             13
#define FUF_CODE             14

#define MAX_POINTER_INDEX    16383

#define MAX_Z_VAL            65535
#define MAX_Z_INDEX          16383
#define MAX_X_VAL            65535
```

```
#define MAX_Y_VAL           65535

#define MAX_DBL             1E+38
#define MAX_UINT            65535
#define ZERO_DEF            1E-5
#define RADIANS_PER_DEGREE  0.01745329266666666666

#define MAX_HASH_INDEX      1501
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPCHL.H                                          */
/*                                                   */
/* PFP Cross-Hatch Line Header                       */
/*****************************************************/

struct hatch_point {
    UINT16              x_val;
    UINT16              y_val;
    UINT16              angle;
    UINT8               enter_flag;
    UINT16              triangle_number;
    struct hatch_point *next_point;
   };
typedef struct hatch_point HATCH_POINT_STRUC;

struct min_max_type {
  UINT16       min_x;
  UINT16       max_x;
  UINT16       min_y;
  UINT16       max_y;
  };

typedef struct min_max_type MIN_MAX_STRUC;

#define NULL_HATCH_POINT (HATCH_POINT_STRUC *) NULL
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPCURE.H                                           */
/*                                                     */
/*******************************************************/

struct cure_hash {
  UINT16            x;
  UINT16            y;
  OUT_SEG_STRUC     *seg_ptr;
  struct cure_hash *last_hash;
  struct cure_hash *next_hash;
  };

typedef struct cure_hash CURE_HASH_STRUC;

#define NULL_CURE_HASH   (CURE_HASH_STRUC *) NULL
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPHI.H                                             */
/*                                                     */
/* PFP Human Interface header file                     */
/*******************************************************/

#define FILE_NAME_LENGTH 100

typedef struct
   {
    float    res_value;
    UINT16   y_flat_fill_spacing;
    UINT16   x_flat_fill_spacing;
    UINT16   y_hatch_spacing;
    UINT16   x_hatch_spacing;
    UINT16   angle_hatch_spacing;
    UINT16   z_spacing;
    UINT16   min_surface_angle;
    UINT16   min_intersect_angle;
    UINT16   axis;
    STRING   z_space_filename [FILE_NAME_LENGTH];
    STRING   internal_filename[FILE_NAME_LENGTH];
    STRING   segfile_name     [FILE_NAME_LENGTH];
    BOOLEAN  binary_input;
    BOOLEAN  debug_mode;
    BOOLEAN  display_on;
    BOOLEAN  centering_off;
    BOOLEAN  section_codes_off;
    BOOLEAN  attrib;
    BOOLEAN  in_mem;
    UINT16    cure;

   } USER_INFO_STRUC;
```

```
#define RES_OPTION                    "-RES"
#define SEGFILE_OPTION                "-SF"
#define HATCH_FILE_OPTION             "-HF"
#define Y_HATCH_SPACE_OPTION          "-HY"
#define X_HATCH_SPACE_OPTION          "-HX"
#define ANGLE_HATCH_SPACE_OPTION      "-HA"
#define YF_HATCH_SPACE_OPTION         "-HFY"
#define XF_HATCH_SPACE_OPTION         "-HFX"
#define HELP_OPTION                   "-H"
#define DISPLAY_OPTION                "-D"
#define Z_SPACE_OPTION                "-ZS"
#define Z_SPACE_FILE_OPTION           "-ZSF"
#define CENTER_OPTION                 "-C"
#define SECTION_CODES_OFF_OPTION      "-NSC"
#define MIN_SURFACE_ANGLE_OPTION      "-MSA"
#define MIN_INTERSECT_ANGLE_OPTION    "-MIA"
#define X_AXIS_OPTION                 "-X"
#define Y_AXIS_OPTION                 "-Y"
#define BINARY_INPUT_OPTION           "-B"
#define DEBUG_OPTION                  "-DEBUG"
#define DISK_OPTION                   "-DISK"
#define ARIES_OPTION                  "-ARIES"
#define INTERNAL_OPTION               "-INTERNAL"
#define CURE_OPTION                   "-CURE"


#define RES_OPTION_CODE                    1
#define SEGFILE_OPTION_CODE                2
#define Y_HATCH_SPACE_OPTION_CODE          3
#define X_HATCH_SPACE_OPTION_CODE          4
#define ANGLE_HATCH_SPACE_OPTION_CODE      5
#define YF_HATCH_SPACE_OPTION_CODE         6
#define XF_HATCH_SPACE_OPTION_CODE         7
#define HELP_OPTION_CODE                   8
#define DEBUG_OPTION_CODE                  9
#define DISPLAY_OPTION_CODE                10
```

```
#define Z_SPACE_OPTION_CODE              11
#define Z_SPACE_FILE_OPTION_CODE         12
#define CENTER_OPTION_CODE               13
#define SECTION_CODES_OFF_OPTION_CODE    14
#define MIN_SURFACE_ANGLE_OPTION_CODE    15
#define MIN_INTERSECT_ANGLE_OPTION_CODE  16
#define X_AXIS_OPTION_CODE               17
#define Y_AXIS_OPTION_CODE               18
#define BINARY_INPUT_OPTION_CODE         19
#define DISK_OPTION_CODE                 20
#define ARIES_OPTION_CODE                21
#define INTERNAL_OPTION_CODE             22
#define CURE_OPTION_CODE                 23
#define UNKNOWN_OPTION_CODE              24

#define DEFAULT_RES_VALUE   1.0
#define DEFAULT_SEGFILE_NAME  ""

#define MAX_FLOAT_LENGTH 9
#define MAX_INT_LENGTH 5
#define MAX_ANGLE 90
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPCHL.H                                          */
/*                                                   */
/* PFP Cross-Hatch Line Header                       */
/*****************************************************/

#define NORMAL_X_POSITIVE  1
#define NORMAL_X_NEGATIVE  2
#define NORMAL_Y_POSITIVE  4
#define NORMAL_Y_NEGATIVE  8
#define NORMAL_Z_POSITIVE  16
#define NORMAL_Z_NEGATIVE  32
#define SPARE1             64
#define SPARE2             128

struct out_seg
   {
    UINT16          x_1;
    UINT16          y_1;
    UINT16          x_2;
    UINT16          y_2;
    UINT8           normal_info;
    struct out_seg *next_seg;

   };

typedef struct out_seg OUT_SEG_STRUC;

#define NULL_SEG         (OUT_SEG_STRUC *) NULL

#define L_TEXT              "L"    /* Layer Header */
#define LB_TEXT           "LB"   /* border */
#define LH_TEXT           "LH"   /* border hatching */
```

```
#define NFDB_TEXT       "NFDB" /* Near-Flat Down skin
                        Border*/
#define NFDH_TEXT       "NFDH" /* Near-Flat Down skin
                        Hatch */
#define NFDF_TEXT       "NFDF" /* Near-Flat Down skin
                        Fill */
#define NFUB_TEXT       "NFUB" /* Near-Flat Up skin
                        Border */
#define NFUH_TEXT       "NFUH" /* Near-Flat Up skin
                        Border */
#define NFUF_TEXT       "NFUF" /* Near-Flat Up skin
                        Fill */
#define FDB_TEXT        "FDB"  /* Flat Down skin Fill
                        */
#define FDH_TEXT        "FDH"  /* Flat Down skin Fill
                        */
#define FDF_TEXT        "FDF"  /* Flat Down skin Fill
                        */
#define FUB_TEXT        "FUB"  /* Flat Down skin Fill
                        */
#define FUH_TEXT        "FUH"  /* Flat Down skin Fill
                        */
#define FUF_TEXT        "FUF"  /* Flat Down skin Fill
                        */
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPTRI.H                                            */
/*                                                     */
/* PFP triangle definition header file                 */
/*******************************************************/

struct in_tri
   {
    float         x1;
    float         y1;
    float         z1;
    float         x2;
    float         y2;
    float         z2;
    float         x3;
    float         y3;
    float         z3;
    UINT8         normal_info;
    struct in_tri *next_input_tri;
   };

typedef struct in_tri INPUT_TRI_STRUC;


struct vector
   {
    float x;
    float y;
    float x_incr;
    float y_incr;
   };


struct vect_tri
```

```
    {
     struct vector   vector_1;
     struct vector   vector_2;
     struct vector   vector_3;
     float           vert1_z;
     float           vert2_z;
     float           vert3_z;
     UINT8           normal_info;
     struct vect_tri *next_tri;
    };

typedef struct vect_tri VECT_TRI_STRUC;

#define NULL_INPUT_TRI  (INPUT_TRI_STRUC *) NULL
#define NULL_VECT_TRI   (VECT_TRI_STRUC *) NULL
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPUTL.H                                          */
/*                                                   */
/* PFP Utility Prototypes                            */
/*****************************************************/

extern float  read_float();
extern double round_dbl();
extern float  round_flt();
extern STRING *mem_alloc();
extern INT16  sign();
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* STORAGE.H - Storage definitions for machine for   */
/* various programs                                  */
/*****************************************************/

typedef unsigned long       UINT32;
typedef long                INT32;
typedef unsigned short      UINT16;
typedef short               INT16;
typedef unsigned char       BOOLEAN;
typedef unsigned char       UINT8;
typedef char                STRING;

#define TRUE                1
#define FALSE               0
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFP.C                                               */
/*                                                     */
/* PFP Main module                                     */
/*******************************************************/

#include <stdio.h>
#include <time.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfphi.h"
#include "pfpseg.h"
#include "pfputl.h"


#define VERSION "3.20"

#define BANNER    "3D Systems Slice Program"
#define COPYRIGHT "Copyright (C) 1988 3D Systems Inc."
/*
   Version 18a - corrected border/skin alignment
           18b - implemented 120 degree hatching
           18c - corrected memory usage reporting
           18d - corrected up/down facets on binary
input
                 slice 1 unit above level
           18e - modified Z spacing table logic for
                              greater number of slice
                            levels
           18f - corrected skin border
           18g - filtered output segments (Point
Segments,
                                 Duplicate Segments)
modified output of
```

'.' to numbers (layer count)

19 - implemented Style I processing changed output block mnemonics

19a - Style IA implemented, having the NFUB block

19b - Implemented segment file post processing

Allowed Z to range up to 65535 with 16384 layers

Added Attributed binary input option

19c - Added holding layer in memory.

Allow upper or lower case for ASCII files

19d - Add -ARIES -X -Y -C options

Delete -NC option

Redo intra-layer vector processing

20 - Implement Normal Code

new version number scheme - V20 also known as V2.30

20a - Modify to work with Unix

232 - Standardize Version number

Implement Style 1B

Correct Var Z Spacing Table Setup

Add check for Z table ranges

Correct Sorting of Hatch and Fill Segments

Correct ASCII input routines to handle '\r's

233 - Allow broken parts in the slice axis

234 - Optional internal triangle database output

Eliminate collapsed line triangles

Align border segments so they will be

eliminated
240 - output slice command line to slice output
file
second major software release
250 - Implement Cross Hatch endpoint counting
Recompute normal vectors of deformed triangles
Eliminate triangle counting
251 - Experimental (No version released)
260 - Recognize Ctrl-Z as EOF in ASCII file input
Change "resd" references (was "scaled")
Error check output to SLI file (I/O, disk full ...)
Only recompute X & Y normal information
261 - Cure Compensation
Use Unsigned integers for Z values in get_min_max_z
Add Copyright notice to all source files and in print out.
Take ARIES option out of help
262 - Continued Cure Compensaation
263 - Correct Sorting to connect first and last segments
264 - Eliminate "small" vectors for cure compensation
Force FUF on first layer to stay on that layer
300 - Clean up warning messeges for release 3.0.
Change Scan to Steep Triangles.
301 - Delete fill borders if no fill

```
            310 - Correct concatination of vectors in
sort
                  enhance logic of sort warning messeges
            311 - IRIS Migration
                  correct sscanf data types
                  iris system not IEEE floats.  Bytes are
                        reversed
                  add -signed to cc command
                  Cure Sorting: in computing normal, use
                        (double) for const
                  General errors
                  correct triangle binary read error to
                        print proper triangle
number in message
                  remove slice parameter output to screen
                        at start of program
            312 - Iris binary file facet fix to reverse 4
                        bytes.
                  changes to program module pfprif.c .
            313 - fixed stray vectors caused by bad
                        normals.
                  changes to program module pfpchl.c .
            315 - added slope and boundary length check
to
                        pfpcure.c to
                  correct for long and short vector
vertex
                        in cure compensation
            316 - correct fatal memory fault problem by
                        removing extra FUF
                  and FUB layers in comb_list in pfp.c.
            317 - correct fatal floating point exception
                        error in make_normal
                  and make_unit_vector in pfpcure.c used
                        in cure compensation
            318 - added resolution scaling to cure
```

```
                                compensation for long &
short vector
                                vertex in pfpcure.c
            319 - corrected angle check to dot product in
                                tpfpcure.c
            320 - added resolution check for short
vectors
                                in tpfpcure.c
                  also provide a different angle if not
                                curing for combining
                  short segments. changes to tpfpcure.c

*/


INPUT_TRI_STRUC *tri_bucket_array     [MAX_Z_INDEX];
OUT_SEG_STRUC   *FU_seg_bucket_array  [MAX_Z_INDEX];
OUT_SEG_STRUC   *FD_seg_bucket_array  [MAX_Z_INDEX];
OUT_SEG_STRUC   *NFU_seg_bucket_array [MAX_Z_INDEX];
OUT_SEG_STRUC   *NFD_seg_bucket_array [MAX_Z_INDEX];
OUT_SEG_STRUC   *post_LB_seg_list        = NULL_SEG;
OUT_SEG_STRUC   *post_LH_seg_list        = NULL_SEG;
OUT_SEG_STRUC   *post_NFDB_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_NFDH_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_NFDF_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_NFUB_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_NFUH_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_NFUF_seg_list      = NULL_SEG;
OUT_SEG_STRUC   *post_FUB_seg_list       = NULL_SEG;
OUT_SEG_STRUC   *post_FUH_seg_list       = NULL_SEG;
OUT_SEG_STRUC   *post_FUF_seg_list       = NULL_SEG;
OUT_SEG_STRUC   *post_FDB_seg_list       = NULL_SEG;
OUT_SEG_STRUC   *post_FDH_seg_list       = NULL_SEG;
OUT_SEG_STRUC   *post_FDF_seg_list       = NULL_SEG;

UINT16                    z_control_table[MAX_Z_INDEX];
unsigned long   post_z_level = 0;
```

```
UINT16          all_min_z, all_max_z, all_min_z_index,
all_max_z_index;

UINT16          post_file_number;
FILE            *post_process_seg_file[2];
STRING          *post_process_filename[2];

long            max_mem_alloc = 0;
long            total_mem_alloc = 0;
long            curr_mem_alloc = 0;

unsigned long   temp_count = 0;

FILE            *message_file;
BOOLEAN         read_message_file = FALSE;

extern    INT16   slice_count;
float resolution;                          /*  global
variable  */

/*******************************************************
   local procedure

   print_user_info

*******************************************************/

static void print_user_info (outfile, user_info)

   FILE            *outfile;
   USER_INFO_STRUC *user_info;

{

   fprintf (outfile, "\n resolution value
..................... %f",
                                        user_info->
```

```
res_value);
   fprintf (outfile, "\n x hatch spacing
..................... %u",
                                        user_info->
x_hatch_spacing);
   fprintf (outfile, "\n y hatch spacing
..................... %u",
                                        user_info->
y_hatch_spacing);
   fprintf (outfile, "\n flat x fill spacing
................. %u",
                                        user_info->
x_flat_fill_spacing);
   fprintf (outfile, "\n flat y fill spacing
................. %u",
                                        user_info->
y_flat_fill_spacing);
   fprintf (outfile, "\n 60/120 degree angle hatch
spac ng .. %u",
                                        user_info->
angle_hatch_spacing);
   fprintf (outfile, "\n min surface angle
................... %u",
                                        user_info->
min_surface_angle);
   fprintf (outfile, "\n min intersect angle
................. %u",
                                        user_info->
min_intersect_angle);
   fprintf (outfile, "\n input file format
................... %s",
                          user_info-> binary_input ?
"binary" : "ascii");
   fprintf (outfile, "\n centering
........................... %s",
                          user_info-> centering_off ?
"off" : "on");
```

```
   fprintf (outfile, "\n output file section codes
..... %s",
                           user_info-> section_codes_off ?
"off" : "on");
   fprintf (outfile, "\n segment output file
.............. %s",
            ((*(user_info-> segfile_name) == '\0') ? "-
not used -"
                                                  :
user_info-> segfile_name));
   if (*(user_info-> internal_filename) > '\0')
      fprintf (outfile, "\n internal triangles output
file ..... %s",
               user_info-> internal_filename);
   if (*(user_info-> z_space_filename) == '\0')

      fprintf (outfile, "\n z spacing
..... %u\n",
                                  user_info-> z_spacing);
   else

      fprintf (outfile, "\n z spacing control file
..... %s\n",
                                  user_info->
z_space_filename);


} /* print_user_info */

/*********************************************************
   local procedure

   get_space_line
*********************************************************/

static void get_space_line (z_space_file, z_change,
new_z_spacing, eof, z)
```

```
   FILE     *z_space_file;
   UINT16   *z_change;
   UINT16   *new_z_spacing;
   BOOLEAN  *eof;
   UINT16   z;

{
   short    read_status;
   short    read_count = 2;
   unsigned int temp1,temp2; /* Must be unsigned in for
scanf */


   read_status = fscanf (z_space_file, "%u %u", &temp1,
&temp2);

   *z_change = (INT16) temp1;
   *new_z_spacing = (INT16) temp2;

   if (read_status != read_count)
      {
       if (read_status == EOF)
          *eof = TRUE;
       else
          {
           fprintf (stderr, "\n error - invalid format
in z spacing file\n");
           exit (SYS_FAILURE);
          }
      }
   else if ((z != MAX_Z_VAL) && (*z_change <= z))
      {
       fprintf (stderr,
          "\n error - z value (%u) in spacing file is
out of sequence \n",
             *z_change);
```

```
        exit (SYS_FAILURE);
       }
   else if ((*z_change < 0) || (*new_z_spacing < 0))
       {
        fprintf (stderr, "\n error - invalid value in z
spacing file");
        exit (SYS_FAILURE);
       }

} /* get_space_line */


/*********************************************************
   local procedure

   get_space_file_data

*********************************************************/

static void get_space_file_data (z_space_file)

   FILE          *z_space_file;

{
   UINT16         new_z_spacing, z_change, z_count = 0,
z;
   BOOLEAN        eof = FALSE;
   unsigned long  zi;
   UINT16         last_z_change, last_new_z_spacing;

   get_space_line (z_space_file, &z_change,
&new_z_spacing, &eof, MAX_Z_VAL);
   while ((!eof) && (z_count < MAX_Z_INDEX)) {
      last_z_change = z_change;
      last_new_z_spacing = new_z_spacing;
      z = z_change;
```

```
      get_space_line (z_space_file, &z_change,
&new_z_spacing, &eof, z);
      zi = last_z_change;
      while (zi < z_change) {
         z_control_table [z_count++] = zi;
         zi += last_new_z_spacing;
      }
   }
   while ((zi < MAX_Z_VAL) && (z_count < MAX_Z_INDEX)) {

      z_control_table [z_count++] = zi;
      zi += last_new_z_spacing;
   }
   while (z_count < MAX_Z_INDEX)
      z_control_table [z_count++] = MAX_Z_VAL;

} /* get_space_file_data */


static build_constant_spacing(user_info)

   USER_INFO_STRUC  *user_info;

{
   UINT16         z_count=0;
   long           zi = 0;

   while ((zi < MAX_Z_VAL) && (z_count < MAX_Z_INDEX)) {

      z_control_table [z_count++] = zi;
      zi += user_info-> z_spacing;
   }
   while (z_count < MAX_Z_INDEX)
      z_control_table [z_count++] = MAX_Z_VAL;

} /* build_constant_spacing */
```

```
/*********************************************************
   local procedure

   set_z_spacing

*********************************************************/


static void set_z_spacing (user_info)

   USER_INFO_STRUC  *user_info;
{     .
   FILE  *z_space_file;

   if (*(user_info-> z_space_filename) != '\0')
      {
       openfile (user_info-> z_space_filename,
&z_space_file, "r");
       get_space_file_data (z_space_file);
      }
   else {
      build_constant_spacing(user_info);
   }

} /* set_z_spacing */

/*********************************************************
   local procedure

   single_min_max_z
*********************************************************/


static void single_min_max_z (a_seg_bucket_array)
   OUT_SEG_STRUC *a_seg_bucket_array[];
{
```

```
    UINT16 i,j;

    for (i = 0; i < MAX_Z_INDEX; i++) {
        j = z_control_table [i];
        if (a_seg_bucket_array [i] != NULL_SEG) {
            if (j > all_max_z) {
              all_max_z_index = i;
              all_max_z = j;
              };
            if (j < all_min_z) {
               all_min_z_index = i;
               all_min_z = j;
            }
        }
    }
} /* single_min_max_z */


static void get_min_max_z ()
{
    UINT16 i,j;

    all_min_z_index = MAX_Z_INDEX-1;
    all_min_z = z_control_table [MAX_Z_INDEX-1];

    for (i = 0; i < MAX_Z_INDEX; i++) {
        j = z_control_table [i];
        if (tri_bucket_array [i] != NULL_INPUT_TRI) {
            all_max_z_index = i;
            all_max_z = j;
            if (j < all_min_z) {
               all_min_z_index = i;
               all_min_z = j;
            }
        }
    }
  single_min_max_z(FU_seg_bucket_array);
```

```
  single_min_max_z(FD_seg_bucket_array);
  single_min_max_z(NFU_seg_bucket_array);
  single_min_max_z(NFD_seg_bucket_array);
}

/*******************************************************
   procedure

   comb_list

          Combines segment list into one big list.  Save
          the type of list in segment structure for
          later extraction.  Does not add point or
          duplicate segments

*******************************************************/

void comb_list(main_list,new_list,list_code)
  OUT_SEG_STRUC **main_list;
  OUT_SEG_STRUC **new_list;
  UINT8          list_code;
  {
  OUT_SEG_STRUC *new_ptr,*scan_ptr,*end_ptr;
  BOOLEAN delete_seg;

  new_ptr = *new_list;
  *new_list = NULL_SEG;
/*  inserting print statements to catch memory fault
error      */

/* fprintf(stderr,"main=%ld new=%ld
list=%d\n",main_list,new_list,list_code); */
/*
        */
  while (new_ptr != NULL_SEG) {
    if ((new_ptr-> x_1 == new_ptr-> x_2) &&
```

```
        (new_ptr-> y_1 == new_ptr-> y_2)) delete_seg =
TRUE;
    else {
      scan_ptr = *main_list;
      end_ptr = NULL_SEG;
      while ((scan_ptr != NULL_SEG) &&
           !((new_ptr->x_1 == scan_ptr->x_1) &&
             (new_ptr->y_1 == scan_ptr->y_1) &&
             (new_ptr->x_2 == scan_ptr->x_2) &&
             (new_ptr->y_2 == scan_ptr->y_2)) &&
           !((new_ptr->x_1 == scan_ptr->x_2) &&
             (new_ptr->y_1 == scan_ptr->y_2) &&
             (new_ptr->x_2 == scan_ptr->x_1) &&
             (new_ptr->y_2 == scan_ptr->y_1))) {
        end_ptr = scan_ptr;
        scan_ptr = scan_ptr-> next_seg;
        }
      if (scan_ptr != NULL_SEG) delete_seg = TRUE;
      else delete_seg = FALSE;
      }
    scan_ptr = new_ptr-> next_seg;
    if (delete_seg)
mem_free(new_ptr,sizeof(OUT_SEG_STRUC));
    else {
      if (*new_list == NULL_SEG) {
        *new_list = new_ptr;
        if (*main_list == NULL_SEG) *main_list =
new_ptr;
        }
      if (end_ptr != NULL_SEG) end_ptr-> next_seg =
new_ptr;
      /* Use "normal_info" to store list type */
      new_ptr-> normal_info = list_code;
      new_ptr-> next_seg = NULL_SEG;
      }
    new_ptr = scan_ptr;
    }
```

```
  };

/*********************************************************
   procedure

   write_post

   Writes a segment list to a temp file for disk
processing.  Save type of list in Segment structure.

*********************************************************/

void write_post(seg_ptr,out_file,type_code)
  OUT_SEG_STRUC *seg_ptr;
  FILE          *out_file;
  UINT16        type_code;
  {
  OUT_SEG_STRUC *temp_ptr;

  while (seg_ptr != NULL_SEG) {
    /* Use "normal_info" to store list type */
    seg_ptr-> normal_info = type_code;
    fwrite(seg_ptr,sizeof(OUT_SEG_STRUC),1,out_file);
    temp_ptr = seg_ptr;
    seg_ptr = seg_ptr-> next_seg;
    mem_free(temp_ptr,sizeof(OUT_SEG_STRUC));
    };
  };


/*********************************************************
   procedure

   read_post

   Read all segment list from prev. layer into post
segment lists.
```

```
**********************************************************/

void read_post(in_file)
  FILE *in_file;
  {
  OUT_SEG_STRUC *seg_ptr;
  OUT_SEG_STRUC *end_LB_seg_list        = NULL_SEG;
  OUT_SEG_STRUC *end_LH_seg_list        = NULL_SEG;
  OUT_SEG_STRUC *end_NFDB_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_NFDH_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_NFDF_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_NFUB_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_NFUH_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_NFUF_seg_list      = NULL_SEG;
  OUT_SEG_STRUC *end_FDB_seg_list       = NULL_SEG;
  OUT_SEG_STRUC *end_FDH_seg_list       = NULL_SEG;
  OUT_SEG_STRUC *end_FDF_seg_list       = NULL_SEG;
  OUT_SEG_STRUC *end_FUB_seg_list       = NULL_SEG;
  OUT_SEG_STRUC *end_FUH_seg_list       = NULL_SEG;
  OUT_SEG_STRUC *end_FUF_seg_list       = NULL_SEG;

  post_LB_seg_list        = NULL_SEG;
  post_LH_seg_list        = NULL_SEG;
  post_NFDB_seg_list      = NULL_SEG;
  post_NFDH_seg_list      = NULL_SEG;
  post_NFDF_seg_list      = NULL_SEG;
  post_NFUB_seg_list      = NULL_SEG;
  post_NFUH_seg_list      = NULL_SEG;
  post_NFUF_seg_list      = NULL_SEG;
  post_FDB_seg_list       = NULL_SEG;
  post_FDH_seg_list       = NULL_SEG;
  post_FDF_seg_list       = NULL_SEG;
  post_FUB_seg_list       = NULL_SEG;
  post_FUH_seg_list       = NULL_SEG;
  post_FUF_seg_list       = NULL_SEG;
```

```
  if (in_file != (FILE *) NULL) {
    seg_ptr = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
    while
((fread(seg_ptr,sizeof(OUT_SEG_STRUC),1,in_file) == 1)
&&
           (feof(in_file) == 0)) {
      seg_ptr-> next_seg = NULL_SEG;
      switch (seg_ptr-> normal_info) { /* "normal_info"
has list type */
        case LB_CODE :
          if (end_LB_seg_list != NULL_SEG)
            end_LB_seg_list-> next_seg = seg_ptr;
          else post_LB_seg_list = seg_ptr;
          end_LB_seg_list = seg_ptr;
          break;
        case LH_CODE :
          if (end_LH_seg_list != NULL_SEG)
            end_LH_seg_list-> next_seg = seg_ptr;
          else post_LH_seg_list = seg_ptr;
          end_LH_seg_list = seg_ptr;
          break;
        case NFDB_CODE :
          if (end_NFDB_seg_list != NULL_SEG)
            end_NFDB_seg_list-> next_seg = seg_ptr;
          else post_NFDB_seg_list = seg_ptr;
          end_NFDB_seg_list = seg_ptr;
          break;
        case NFDH_CODE :
          if (end_NFDH_seg_list != NULL_SEG)
            end_NFDH_seg_list-> next_seg = seg_ptr;
          else post_NFDH_seg_list = seg_ptr;
          end_NFDH_seg_list = seg_ptr;
          break;
        case NFDF_CODE :
          if (end_NFDF_seg_list != NULL_SEG)
            end_NFDF_seg_list-> next_seg = seg_ptr;
```

```
    else post_NFDF_seg_list = seg_ptr;
    end_NFDF_seg_list = seg_ptr;
    break;
  case NFUB_CODE :
    if (end_NFUB_seg_list != NULL_SEG)
      end_NFUB_seg_list-> next_seg = seg_ptr;
    else post_NFUB_seg_list = seg_ptr;
    end_NFUB_seg_list = seg_ptr;
    break;
  case NFUH_CODE :
    if (end_NFUH_seg_list != NULL_SEG)
      end_NFUH_seg_list-> next_seg = seg_ptr;
    else post_NFUH_seg_list = seg_ptr;
    end_NFUH_seg_list = seg_ptr;
    break;
  case NFUF_CODE :
    if (end_NFUF_seg_list != NULL_SEG)
      end_NFUF_seg_list-> next_seg = seg_ptr;
    else post_NFUF_seg_list = seg_ptr;
    end_NFUF_seg_list = seg_ptr;
    break;
  case FDB_CODE :
    if (end_FDB_seg_list != NULL_SEG)
      end_FDB_seg_list-> next_seg = seg_ptr;
    else post_FDB_seg_list = seg_ptr;
    end_FDB_seg_list = seg_ptr;
    break;
  case FDH_CODE :
    if (end_FDH_seg_list != NULL_SEG)
      end_FDH_seg_list-> next_seg = seg_ptr;
    else post_FDH_seg_list = seg_ptr;
    end_FDH_seg_list = seg_ptr;
    break;
  case FDF_CODE :
    if (end_FDF_seg_list != NULL_SEG)
      end_FDF_seg_list-> next_seg = seg_ptr;
    else post_FDF_seg_list = seg_ptr;
```

```
            end_FDF_seg_list = seg_ptr;
            break;
          case FUB_CODE :
            if (end_FUB_seg_list != NULL_SEG)
              end_FUB_seg_list-> next_seg = seg_ptr;
            else post_FUB_seg_list = seg_ptr;
            end_FUB_seg_list = seg_ptr;
            break;
          case FUH_CODE :
            if (end_FUH_seg_list != NULL_SEG)
              end_FUH_seg_list-> next_seg = seg_ptr;
            else post_FUH_seg_list = seg_ptr;
            end_FUH_seg_list = seg_ptr;
            break;
          case FUF_CODE :
            if (end_FUF_seg_list != NULL_SEG)
              end_FUF_seg_list-> next_seg = seg_ptr;
            else post_FUF_seg_list = seg_ptr;
            end_FUF_seg_list = seg_ptr;
            break;
          default :
            fprintf(stderr,"Unknow Section Code
(read_post) %u\n",seg_ptr->normal_info);
            exit(1);
            break;
          };
        seg_ptr = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
        };
      mem_free(seg_ptr,sizeof(OUT_SEG_STRUC));
      };
    };



/*******************************************************
   local procedure
```

```
   process_output_section

*************************************************************/


static void process_output_section (LB_seg_list,
z_index,
                                    z, section_count,
user_info, main_seg_file)


   OUT_SEG_STRUC     **LB_seg_list;
   UINT16            z_index,z,section_count;
   USER_INFO_STRUC   *user_info;
   FILE              *main_seg_file;

{
   OUT_SEG_STRUC *master_seg_list          = NULL_SEG;
   OUT_SEG_STRUC *LH_seg_list              = NULL_SEG;
   OUT_SEG_STRUC *NFUH_seg_list            = NULL_SEG;
   OUT_SEG_STRUC *NFUF_seg_list            = NULL_SEG;
   OUT_SEG_STRUC *NFDH_seg_list            = NULL_SEG;
   OUT_SEG_STRUC *NFDF_seg_list            = NULL_SEG;
   OUT_SEG_STRUC *FDH_seg_list             = NULL_SEG;
   OUT_SEG_STRUC *FDF_seg_list             = NULL_SEG;
   OUT_SEG_STRUC *FUH_seg_list             = NULL_SEG;
   OUT_SEG_STRUC *FUF_seg_list             = NULL_SEG;

   OUT_SEG_STRUC *temp_ptr;
   INT16         loop_count = 0;
   BOOLEAN       do_hatch_flag,do_fill_flag;
   UINT16        last_z_level;

 if (post_z_level > (unsigned long) MAX_Z_VAL) {
   pfp_delete_seg_list(LB_seg_list);
   pfp_delete_seg_list(&post_LB_seg_list);
```

```
    pfp_delete_seg_list(&post_LH_seg_list);
    pfp_delete_seg_list(&post_NFDB_seg_list);
    pfp_delete_seg_list(&post_NFDH_seg_list);
    pfp_delete_seg_list(&post_NFDF_seg_list);
    pfp_delete_seg_list(&post_NFUB_seg_list);
    pfp_delete_seg_list(&post_NFUH_seg_list);
    pfp_delete_seg_list(&post_NFUF_seg_list);
    pfp_delete_seg_list(&post_FDB_seg_list);
    pfp_delete_seg_list(&post_FDH_seg_list);
    pfp_delete_seg_list(&post_FDF_seg_list);
    pfp_delete_seg_list(&post_FUB_seg_list);
    pfp_delete_seg_list(&post_FUH_seg_list);
    pfp_delete_seg_list(&post_FUF_seg_list);
    }
  else {
    last_z_level = (UINT16) post_z_level;
    do_hatch_flag = (user_info-> y_hatch_spacing) ||
                    (user_info-> x_hatch_spacing) ||
                    (user_info-> angle_hatch_spacing);

    do_fill_flag  = (user_info-> y_flat_fill_spacing) ||

                    (user_info-> x_flat_fill_spacing);


    if (!do_fill_flag) {
      pfp_delete_seg_list(&FU_seg_bucket_array[z_index]);
      pfp_delete_seg_list(&FD_seg_bucket_array[z_index]);

pfp_delete_seg_list(&NFU_seg_bucket_array[z_index]);

pfp_delete_seg_list(&NFD_seg_bucket_array[z_index]);
     }
    else {
   pfp_remove_duplicate_edges(&FU_seg_bucket_array
     [z_index]);
```

```
    pfp_remove_duplicate_edges(&FD_seg_bucket_array
      [z_index]);
    pfp_remove_duplicate_edges(&NFU_seg_bucket_array
      [z_index]);
    pfp_remove_duplicate_edges(&NFD_seg_bucket_array
      [z_index]);
       };

pfp_remove_duplicate_edges(LB_seg_list);

    cure_sort(LB_seg_list,user_info,section_count,z,0);
cure_sort(&FU_seg_bucket_array[z_index],user_info,
    section_count,z,1);
cure_sort(&FD_seg_bucket_array[z_index],user_info,
    section_count,z,1);
cure_sort(&NFU_seg_bucket_array[z_index],user_info,
    section_count,z,1);
cure_sort(&NFD_seg_bucket_array[z_index],user_info,
    section_count,z,1);

/* added flags to check for other than layer boundaries
*/

    if (user_info->cure>0) {
      cure_compensate(*LB_seg_list,user_info,0);
     cure_compensate(FU_seg_bucket_array[z_index],
       user_info,1);
     cure_compensate(FD_seg_bucket_array[z_index],
       user_info,1);
     cure_compensate(NFU_seg_bucket_array[z_index],
       user_info,1);
     cure_compensate(NFD_seg_bucket_array[z_index],
       user_info,1);
      };

    if (do_hatch_flag) {
      pfp_get_hatch_lines (*LB_seg_list, &LH_seg_list,
```

```
                              user_info-> x_hatch_spacing,
                              user_info-> y_hatch_spacing,
                              user_info->
angle_hatch_spacing,
                              user_info->
min_intersect_angle,FALSE);
  pfp_get_hatch_lines (NFD_seg_bucket_array[z_index],
    &NFDH_seg_list,
                              user_info-> x_hatch_spacing,
                              user_info-> y_hatch_spacing,
                              user_info->
angle_hatch_spacing,
                              user_info->
min_intersect_angle,TRUE);
     );

/* add the min_intersect_angle to the 4 functions below
- before was zero */

   if (do_fill_flag) {
    pfp_get_hatch_lines (FU_seg_bucket_array[z_index],
      &FUF_seg_list,
                              user_info->
          x_flat_fill_spacing,
                              user_info->
                              y_flat_fill_spacing, 0,
                              user_info->
                              min_intersect_angle, TRUE);
    pfp_get_hatch_lines (FD_seg_bucket_array[z_index],
      &FDF_seg_list,
                              user_info->
                              x_flat_fill_spacing,
                              user_info->
                              y_flat_fill_spacing, 0,
                              user_info->
                              min_intersect_angle, TRUE);
```

```
    pfp_get_hatch_lines (NFU_seg_bucket_array[z_index],
      &NFUF_seg_list,
                          user_info->
x_flat_fill_spacing,
                          user_info->
                          y_flat_fill_spacing, 0,
                          user_info->
                          min_intersect_angle, TRUE);
    pfp_get_hatch_lines (NFD_seg_bucket_array[z_index],
      &NFDF_seg_list,
                          user_info->
x_flat_fill_spacing,
                          user_info->
                          y_flat_fill_spacing, 0,
                          user_info->
                          min_intersect_angle, TRUE);
     );


  if (main_seg_file != (FILE *) NULL) {
    if (!user_info-> in_mem) {
    post_process_seg_file[post_file_number] =
      fopen(post_process_filename[post_file_number],"wb"
      );
      write_post(*LB_seg_list, post_process_seg_file
        [post_file_number],LB_CODE);
      write_post(LH_seg_list, post_process_seg_file
        [post_file_number], LH_CODE);
      write_post(NFD_seg_bucket_array[z_index],
        post_process_seg_file[post_file_number],
        NFDB_CODE);
      write_post(NFDH_seg_list, post_process_seg_file
        [post_file_number],NFDH_CODE);
      write_post(NFDF_seg_list, post_process_seg_file
        [post_file_number],NFDF_CODE);
```

```
        write_post(NFU_seg_bucket_array[z_index],
          post_process_seg_file[post_file_number],
          NFUB_CODE);
        write_post(NFUH_seg_list, post_process_seg_file
          [post_file_number],NFUH_CODE);
        write_post(NFUF_seg_list, post_process_seg_file
          [post_file_number],NFUF_CODE);
/*
        write_post(FD_seg_bucket_array[z_index],
          post_process_seg_file[post_file_number],FDB_CODE
          );
        write_post(FDH_seg_list, post_process_seg_file
          [post_file_number],FDH_CODE);
*/
        write_post(FDF_seg_list, post_process_seg_file
          [post_file_number],FDF_CODE);
/*
        write_post(FU_seg_bucket_array[z_index],
          post_process_seg_file[post_file_number],FUB_CODE
          );
        write_post(FUH_seg_list, post_process_seg_file
          [post_file_number],FUH_CODE);
        write_post(FUF_seg_list, post_process_seg_file
          [post_file_number],FUF_CODE);
*/
         fclose(post_process_seg_file[post_file_number]);

         post_file_number = (++post_file_number)%2;
         post_process_seg_file[post_file_number] =
fopen(post_process_filename[post_file_number],"rb");

read_post(post_process_seg_file[post_file_number]);
         fclose(post_process_seg_file[post_file_number]);
         }

     comb_list(&master_seg_list,&post_LB_seg_list,
LB_CODE);
```

```
      comb_list(&master_seg_list,&post_LH_seg_list,
LH_CODE);
      comb_list(&master_seg_list,&post_NFDB_seg_list,
NFDB_CODE);
      comb_list(&master_seg_list,&post_NFDH_seg_list,
NFDH_CODE);
      comb_list(&master_seg_list,&post_NFDF_seg_list,
NFDF_CODE);
      comb_list(&master_seg_list,&post_NFUB_seg_list,
NFUB_CODE);
      comb_list(&master_seg_list,&post_NFUF_seg_list,
NFUF_CODE);
      comb_list(&master_seg_list,&post_FDB_seg_list,
FDB_CODE);
      comb_list(&master_seg_list,&post_FDF_seg_list,
FDF_CODE);
      if (z_index!=all_min_z_index) {
        comb_list(&master_seg_list,&FU_seg_bucket_array
[z_index],FUB_CODE);
        if (post_FUB_seg_list!=NULL_SEG)
          comb_list(&master_seg_list,&post_FUB_seg_list,
FUB_CODE);
        comb_list(&master_seg_list,&FUF_seg_list,
FUF_CODE);
        if (post_FUF_seg_list!=NULL_SEG)
          comb_list(&master_seg_list,&post_FUF_seg_list,
FUF_CODE);
        post_FUB_seg_list = NULL_SEG;
        post_FUF_seg_list = NULL_SEG;
        }
       lse {
        post_FUB_seg_list  =
FU_seg_bucket_array[z_index];
        post_FUF_seg_list  = FUF_seg_list;
        FU_seg_bucket_array[z_index] = NULL_SEG;
        FUF_seg_list = NULL_SEG;
        );
```

```
      if (master_seg_list != NULL_SEG) {

pfp_print_seg_list(&post_LB_seg_list,last_z_level,
main_seg_file,LB_CODE,user_info);

pfp_print_seg_list(&post_LH_seg_list,last_z_level,
main_seg_file,LH_CODE,user_info);
        pfp_print_seg_list(&post_NFDB_seg_list,
last_z_level,main_seg_file,NFDB_CODE,user_info);
        pfp_print_seg_list(&post_NFDH_seg_list,
last_z_level,main_seg_file,NFDH_CODE,user_info);
        pfp_print_seg_list(&post_NFUB_seg_list,
last_z_level,main_seg_file,NFUB_CODE,user_info);

pfp_print_seg_list(&post_FDB_seg_list,last_z_level,
main_seg_file, FDB_CODE,user_info);

pfp_print_seg_list(&post_FDF_seg_list,last_z_level,
main_seg_file, FDF_CODE,user_info);
        pfp_print_seg_list(&post_NFDF_seg_list,
last_z_level,main_seg_file,NFDF_CODE,user_info);
        pfp_print_seg_list(&post_NFUF_seg_list,
last_z_level,main_seg_file,NFUF_CODE,user_info);
        pfp_print_seg_list(&FU_seg_bucket_array[z_index],
last_z_level,main_seg_file, FUB_CODE, user_info);
        pfp_print_seg_list(&FUF_seg_list, last_z_level,
main_seg_file, FUF_CODE, user_info);
        fflush(main_seg_file);
        }

      post_z_level = (unsigned long) z;

      if (user_info-> in_mem) {
        post_LB_seg_list   = *LB_seg_list;
        post_LH_seg_list   = LH_seg_list;
```

```
        post_NFDB_seg_list =
NFD_seg_bucket_array[z_index];
        post_NFDH_seg_list = NFDH_seg_list;
        post_NFUB_seg_list =
NFU_seg_bucket_array[z_index];
        post_FDB_seg_list  =
FD_seg_bucket_array[z_index];
        post_FDF_seg_list  = FDF_seg_list;
        post_NFDF_seg_list = NFDF_seg_list;
        post_NFUF_seg_list = NFUF_seg_list;
        }
      }
    FD_seg_bucket_array[z_index] = NULL_SEG;
    FU_seg_bucket_array[z_index] = NULL_SEG;
    NFD_seg_bucket_array[z_index] = NULL_SEG;
    NFU_seg_bucket_array[z_index] = NULL_SEG;
    };
} /* process_output_section */



/*******************************************************
    local procedure

    generate_cross_sections

*******************************************************/


static void generate_cross_sections (seg_file,
user_info)

    FILE              *seg_file;
```

```
   USER_INFO_STRUC  *user_info;

{
   INPUT_TRI_STRUC **tri_bucket_ptr;
   OUT_SEG_STRUC   *border_seg_list;
   VECT_TRI_STRUC  *z_active_list;
   UINT16          zi, zon, section_count = 0;


   z_active_list = NULL_VECT_TRI;

   get_min_max_z ();  /* sets all_'s */
   zi = all_min_z_index;
   do {
      zon = z_control_table [zi];
      section_count++;
      if (!user_info-> display_on) {
        fprintf(stderr," generating cross section %u
level %.3g (%u)\r",
           section_count,zon/user_info->res_value,zon);
        fflush(stderr);
        };
      tri_bucket_ptr = &tri_bucket_array [zi];
      border_seg_list = NULL_SEG;
      pfp_update_z_active_list (tri_bucket_ptr,
&border_seg_list, &z_active_list);
      pfp_process_z_active_list (&z_active_list, zon,
&border_seg_list);

      process_output_section (&border_seg_list, zi, zon,
section_count, user_info, seg_file);
      fflush(message_file);
      zi++;
   } while ((zi <= all_max_z_index) || (z_active_list !=
NULL_VECT_TRI));

  fprintf(stderr,"\n");
```

```
} /* generate_cross_sections */


/*******************************************************

   local procedure

   init_slice_output_file

*******************************************************/


static void init_slice_output_file (user_info, seg_file)

   USER_INFO_STRUC *user_info;
   FILE *seg_file;

{
   fprintf(seg_file,"!3DSYS SLA-1 SLICE FILE\n");
   fprintf(seg_file,"!SLICE= -VER %s",VERSION);
   fprintf(seg_file," %s %f",RES_OPTION,user_info->
res_value);
   fprintf(seg_file," %s
%u",X_HATCH_SPACE_OPTION,user_info-> x_hatch_spacing);
   fprintf(seg_file," %s
%u",Y_HATCH_SPACE_OPTION,user_info-> y_hatch_spacing);
   fprintf(seg_file," %s
%u",XF_HATCH_SPACE_OPTION,user_info->
x_flat_fill_spacing);
```

```
   fprintf(seg_file," %s
%u",YF_HATCH_SPACE_OPTION,user_info->
y_flat_fill_spacing);
   fprintf(seg_file," %s
%u",ANGLE_HATCH_SPACE_OPTION,user_info->
angle_hatch_spacing);
   fprintf(seg_file," %s
%u",MIN_SURFACE_ANGLE_OPTION,user_info->
min_surface_angle);
   fprintf(seg_file," %s
%u",MIN_INTERSECT_ANGLE_OPTION,user_info->
min_intersect_angle);
   if (*(user_info-> z_space_filename) > '\0')
      fprintf(seg_file," %s
%s",Z_SPACE_FILE_OPTION,user_info-> z_space_filename);
   else
      fprintf(seg_file," %s
%u",Z_SPACE_OPTION,user_info-> z_spacing);
   if (*(user_info-> internal_filename) > '\0')
      fprintf(seg_file," %s
%s",INTERNAL_OPTION,user_info-> internal_filename);
   if (user_info-> axis == 1)
      fprintf(seg_file," %s",X_AXIS_OPTION);
   if (user_info-> axis == 2)
      fprintf(seg_file," %s",Y_AXIS_OPTION);
   if (user_info-> binary_input)
      fprintf(seg_file," %s",BINARY_INPUT_OPTION);
   if (user_info-> centering_off == FALSE)
      fprintf(seg_file," %s",CENTER_OPTION);
   if (user_info-> section_codes_off == TRUE)
      fprintf(seg_file," %s",SECTION_CODES_OFF_OPTION);
   if (user_info-> in_mem == FALSE)
      fprintf(seg_file," %s",DISK_OPTION);
   if (user_info-> attrib == FALSE)
      fprintf(seg_file," %s",ARIES_OPTION);
   if (user_info-> cure != 0)
```

```
      fprintf(seg_file," %s %u",CURE_OPTION,user_info->
cure);
   fprintf(seg_file,"\n");
} /* init_slice_output_file */



/*********************************************************

   Main procedure for PFP

*********************************************************/


main (argc, argv)

   INT16  argc;
   STRING *argv[];

{
   USER_INFO_STRUC user_info;
   FILE            *seg_file;
   STRING          start_time [100],end_time[100];
   STRING
line[80],file_str0[16],file_str1[16],filename[80];
   INT16           i;
   OUT_SEG_STRUC   *dummy_seg;


   get_timestamp (start_time);

   fprintf (stderr, "\n %s      Version %s       %s",
                    BANNER, VERSION, start_time);
```

```
   fprintf (stderr, "\n
%s\n",COPYRIGHT);

   post_file_number = 0;
   temp_count = 0;

   pfp_get_user_info (argc, argv, &user_info);

   resolution = user_info.res_value;

/*   fprintf(stderr,"resolution = %f",resolution);
*/


/*   print_user_info (stderr, &user_info); */
   set_z_spacing (&user_info);
   if (*(user_info.segfile_name) != '\0')
     openfile (user_info .segfile_name, &seg_file, "w");

   else seg_file = stdout;

   i=0;
   while ((i<strlen(user_info.segfile_name)) &&
          (user_info.segfile_name[i]!='.')) {
     filename[i]=user_info.segfile_name[i];
     i++;
     };
   filename[i+1]='\0';
   strcpy(line,filename);
   strcat(line,".msg");

   fprintf(stderr,"\n\n                              Output
File Name : '%s'\n",user_info.segfile_name);
   fprintf(stderr,"                             Message
File Name: '%s'\n",line);

   openfile (line, &message_file, "w");
```

```
   fprintf(message_file,"\nSlice Message File for
'%s'\n",
            user_info.segfile_name);
   fprintf(message_file," Started %s\n\n",start_time);

   init_slice_output_file (&user_info,message_file);
   fprintf(message_file,"\n");
   fflush(message_file);

   pfp_get_input_database (stdin, &user_info);

   if (user_info .display_on)
      init_graphics ();

   post_process_filename[0] = file_str0;
   post_process_filename[1] = file_str1;
   if (!user_info.in_mem) for (i=0; i<2; i++) {
     strcpy(post_process_filename[i],"SLICEXXXXXX");
     mktemp(post_process_filename[i]);
     post_process_seg_file[i] =
fopen(post_process_filename[i],"wb");
     fclose(post_process_seg_file[i]);
     }

   init_slice_output_file (&user_info,seg_file);

   generate_cross_sections (seg_file, &user_info);

   if (user_info .display_on)
      end_graphics ();

   post_z_level += ((unsigned long) MAX_Z_VAL + 1);
   dummy_seg = NULL_SEG;
   process_output_section(&dummy_seg,1,0,&user_info,
seg_file);
   if (!user_info.in_mem) for (i=0; i<2; i++) {
     strcpy(line,"rm ");
```

```
      strcat(line,post_process_filename[i]);
      system(line);
      };

   fprintf (stderr, "\n maximum memory allocation : %ld
bytes\n",
                        max_mem_alloc);
   fprintf (stderr, " total memory allocation    : %ld
bytes\n",
                        total_mem_alloc);
   if (curr_mem_alloc != 0)
     fprintf(stderr," warning: %ld bytes not
freed\n",curr_mem_alloc);
   get_timestamp (end_time);
   fprintf (stderr, "\n");
   fprintf (stderr, " Start Time: %s",start_time);
   fprintf (stderr, "   End Time: %s",end_time);

   if (read_message_file) fprintf(stderr,"\n
  **** Read Message file!  You have Warnings!
****\n\n");
   fprintf(message_file,"\nSlice Ended %s\n",end_time);
   fclose(message_file);
   exit(SYS_SUCCESS);
}
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPCHL.C                                          */
/*                                                   */
/* PFP Cross-Hatch Line Module                       */
/*****************************************************/

#include <stdio.h>
#include <math.h>

#include "pfp.h"
#include "pfpseg.h"
#include "pfpchl.h"
#include "pfputl.h"


#define IN_FLAG           1
#define OUT_FLAG          2
double       hatch_angle;
double       hatch_m;
double       hatch_b_value;
double       hatch_b_spacing;
double       hatch_bottom;
UINT16       hatch_max_index;
BOOLEAN      elim_flat_tri;
INT16        incount;

static HATCH_POINT_STRUC *hatch_point_table
[MAX_POINTER_INDEX];
static BOOLEAN hatch_clear = FALSE;

static HATCH_POINT_STRUC *scan_x(point)
  HATCH_POINT_STRUC **point;
  {
  HATCH_POINT_STRUC *back_ptr = NULL_HATCH_POINT;
```

```
  UINT16              min_angle,curr_x;
  UINT8               flag;

  if (*point != NULL_HATCH_POINT) {
    incount = 0;
    min_angle = MAX_UINT;
    curr_x = (*point)->x_val;
    do {
      if ((*point)->enter_flag&IN_FLAG) incount++;
      else incount--;
      if ((*point)->angle < min_angle) min_angle =
(*point)->angle;
      back_ptr = *point;
      *point = (*point)->next_point;
    } while ((*point != NULL_HATCH_POINT) && (curr_x ==
(*point)->x_val));
    back_ptr->angle = min_angle;
    };
  return(back_ptr);
  };

/*******************************************************

   local procedure

   locate_segment_points

   Returns two hatch line endpoints from hatch point
table of a valid hatch line.  Looks for one "in" and one
"out" point to make segment.  Uses in/out area count to
evaluate critical points.

*******************************************************/

static void locate_segment_points (hatch_index,point1,
point2)
  UINT16              hatch_index;
```

```
  HATCH_POINT_STRUC **point1;
  HATCH_POINT_STRUC **point2;
  {
  HATCH_POINT_STRUC *scan_ptr,*back_ptr;

  scan_ptr = hatch_point_table [hatch_index];

  /* Find first IN point */
  do
    *point1 = scan_x(&scan_ptr);
  while ((*point1 != NULL_HATCH_POINT) && (incount <=
0));

  /* Look for next X where only OUT points exist */
  if (*point1 != NULL_HATCH_POINT) {
    *point2 = NULL_HATCH_POINT;
    while ((scan_ptr != NULL_HATCH_POINT) && (*point2 ==
NULL_HATCH_POINT)) {
      *point2 = scan_x(&scan_ptr);
      if (incount >= 0) *point2 = NULL_HATCH_POINT;
      };
    };

  if ((*point1 == NULL_HATCH_POINT) || (*point2 ==
NULL_HATCH_POINT)) {
    *point1 = NULL_HATCH_POINT;
    *point2 = NULL_HATCH_POINT;
    while (hatch_point_table[hatch_index] !=
NULL_HATCH_POINT) {
      back_ptr = hatch_point_table[hatch_index];
      hatch_point_table[hatch_index] =
back_ptr->next_point;
      mem_free (back_ptr,sizeof(HATCH_POINT_STRUC));
      };
    }
  else
```

```
    while (hatch_point_table[hatch_index] !=
(*point2)->next_point) {
      back_ptr = hatch_point_table[hatch_index];
      hatch_point_table[hatch_index] =
back_ptr->next_point;
      if ((back_ptr != *point1) && (back_ptr !=
*point2))
        mem_free (back_ptr,sizeof(HATCH_POINT_STRUC));
      };

); /* locate_segment_points */


/*********************************************************

    exchange_xy_normal_info

    Exchanges the X and Y normal information in a
normal_info byte.
     Used for 90 degree rotation (swapping X & Y values)

*********************************************************/

static void exchange_xy_normal_info(normal_info)
  UINT8 *normal_info;
  {
  UINT8 new_norm;

  new_norm = *normal_info&(255 - NORMAL_X_NEGATIVE -
NORMAL_X_POSITIVE
                              - NORMAL_Y_NEGATIVE -
NORMAL_Y_POSITIVE);
  if (*normal_info&NORMAL_X_POSITIVE) new_norm |=
NORMAL_Y_POSITIVE;
  if (*normal_info&NORMAL_X_NEGATIVE) new_norm |=
NORMAL_Y_NEGATIVE;
```

```
  if (*normal_info&NORMAL_Y_POSITIVE) new_norm |=
NORMAL_X_POSITIVE;
  if (*normal_info&NORMAL_Y_NEGATIVE) new_norm |=
NORMAL_X_NEGATIVE;
  *normal_info = new_norm;
  };



/*********************************************************

    compute_b_value

    Finds B intersect of hatch line on or below a point

*********************************************************/

static double compute_b_value(x,y)
  UINT16 x,y;
  {

  return((double) (y - hatch_m * x));
  };

/*********************************************************

    compute_b_index

    Returns the index value in the hatch point table for
the hatch B value.

*********************************************************/

static UINT16 compute_b_index(b)
  double b;
  {
```

```
  double round_dbl();

  return((UINT16)
round_dbl((b-hatch_bottom)/hatch_b_spacing));
  );*/


/*******************************************************

    compute_low_b_index

    Returns the index value in the hatch point table on
or below for the hatch
     B value.

*******************************************************

static UINT16 compute_low_b_index(b)
  double b;
  {
  double round_dbl();

  return((UINT16)
floor((b-hatch_bottom)/hatch_b_spacing));
  );*/


/*******************************************************


    get_segment

    Goes threw hatch point list (locate_seg_points) and
adds a hatch line to segment list if one is found.

*******************************************************/
```

```
static void get_segment (hatch_index, hatch_list,
min_intersect_angle,
                         hatch_line_found)

   UINT16             hatch_index;
   OUT_SEG_STRUC      **hatch_list;
   UINT16             min_intersect_angle;
   BOOLEAN            *hatch_line_found;
   {
   OUT_SEG_STRUC      *seg_ptr;
   HATCH_POINT_STRUC *point1_ptr, *point2_ptr;

   *hatch_line_found = FALSE;

   locate_segment_points (hatch_index, &point1_ptr,
&point2_ptr);

   if ((point1_ptr != NULL_HATCH_POINT) &&
       (point1_ptr-> angle >= min_intersect_angle) &&
       (point2_ptr-> angle >= min_intersect_angle)) {
     seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
     seg_ptr-> x_1 = point1_ptr-> x_val;
     seg_ptr-> x_2 = point2_ptr-> x_val;
     seg_ptr-> y_1 = point1_ptr-> y_val;
     seg_ptr-> y_2 = point2_ptr-> y_val;
     seg_ptr-> next_seg = *hatch_list;
     *hatch_list = seg_ptr;
     *hatch_line_found = TRUE;

/*  add check for amount of current memory used during
skin fills    */

     }

   if (point1_ptr != NULL_HATCH_POINT) {
```

```
    mem_free (point1_ptr,sizeof(HATCH_POINT_STRUC));
    mem_free (point2_ptr,sizeof(HATCH_POINT_STRUC));
    };

} /* get_segment */



/*********************************************************

   local procedure

   add_point_to_table

   Adds a hatch point to hatch point table

*********************************************************/

static void add_point_to_table
(x,y,new_angle,new_enter_flag)
  UINT16            x,y;
  INT16             new_angle;
  UINT8             new_enter_flag;
  (
  HATCH_POINT_STRUC *front_ptr,
*back_ptr,*new_point_ptr;
  UINT16            hatch_index;
  BOOLEAN           add_flag;

  hatch_index = floor((hatch_b_value - hatch_bottom +
hatch_b_spacing/2)
                      /hatch_b_spacing);

  if (hatch_index >= MAX_POINTER_INDEX) {
    fprintf (stderr, "\n error - number of hatch lines
exceeds limit of %u\n",(UINT16) MAX_POINTER_INDEX);
```

```
      exit (SYS_FAILURE);
      };

  front_ptr = hatch_point_table [hatch_index];
  back_ptr  = NULL_HATCH_POINT;

/* Search List for X value */
  while ((front_ptr != NULL_HATCH_POINT) &&
         (front_ptr-> x_val < x)) {
    back_ptr  = front_ptr;
    front_ptr = front_ptr-> next_point;
    };

  new_point_ptr = (HATCH_POINT_STRUC *)
mem_alloc(sizeof(HATCH_POINT_STRUC));
  if (back_ptr == NULL_HATCH_POINT) {
    new_point_ptr->next_point =
hatch_point_table[hatch_index];
    hatch_point_table[hatch_index] = new_point_ptr;
    }
  else {
    new_point_ptr->next_point = front_ptr;
    back_ptr->next_point = new_point_ptr;
    };
  new_point_ptr-> x_val = x;
  new_point_ptr-> y_val = y;
  new_point_ptr-> angle = new_angle;
  new_point_ptr-> enter_flag = new_enter_flag;
  if (hatch_index > hatch_max_index) hatch_max_index =
hatch_index;

  }; /* add_point_to_table */

/*******************************************************

    compute_line
```

```
    Compute M and B for line between two points

*****************************************************/

static void compute_line(x1,y1,x2,y2,m,b)
  UINT16       x1,y1;
  UINT16       x2,y2;
  double   *m;
  double   *b;
  {
  if (x1 == x2) {
    *m = (double) MAX_DBL;
    *b = (double) x1;
    }
  else {
    *m = ((double) y2 - y1)/((double) x2 - x1);
    *b = (double) y1 - (*m)*(double) x1;
    };
  };

/*******************************************************

    get_hatch_bottom

    Finds lowest b a segment point and a hatch line will
generate so we know what to start point array at.

*******************************************************/

static get_hatch_bottom(seg_list,swap_xy)
```

```
  OUT_SEG_STRUC *seg_list;
  BOOLEAN       swap_xy;
  {
  double b;

  hatch_bottom = MAX_DBL;

  while (seg_list != NULL_SEG) {
    if (swap_xy) b =
compute_b_value(seg_list->y_1,seg_list->x_1);
    else b =
compute_b_value(seg_list->x_1,seg_list->y_1);
    if (b < hatch_bottom) hatch_bottom = b;
    if (swap_xy) b =
compute_b_value(seg_list->y_2,seg_list->x_2);
    else b =
compute_b_value(seg_list->x_2,seg_list->y_2);
    if (b < hatch_bottom) hatch_bottom = b;
    seg_list = seg_list-> next_seg;
    }
  hatch_bottom = ceil(hatch_bottom/hatch_b_spacing) *
hatch_b_spacing;
  };



/*******************************************************

    compute_intersect

    Finds intersect point of hatch line and passed line.

*******************************************************/

static BOOLEAN compute_intersect(m,b,min_max_info,x,y)
  double        m,b;
```

```
  MIN_MAX_STRUC *min_max_info;
  UINT16        *x,*y;
  {
  double       dx,dy;
  double       round_dbl();

  if (m != hatch_m) {
    if (m == MAX_DBL)  dx = b;
    else dx = (b-hatch_b_value)/(hatch_m-m);
    dy = round_dbl(hatch_m*dx + hatch_b_value);
    dx = round_dbl(dx);
    if ((dx >= min_max_info->min_x) && (dx <=
min_max_info->max_x) &&
        (dy >= min_max_info->min_y) && (dy <=
min_max_info->max_y)) {
      *x = (UINT16) (dx);
      *y = (UINT16) (dy);
      return(TRUE);
      }
    };
  return(FALSE);
  };


/*******************************************************

    get_hatch_points

    Gets hatch intersect points for border segment and
stores into hatch level

*******************************************************/

static void get_hatch_points(seg_ptr,swap_xy)
  OUT_SEG_STRUC *seg_ptr;
  BOOLEAN       swap_xy;
```

```
{
UINT16          x1,y1,x2,y2;
UINT16          hatch_x,hatch_y;
double          seg_m,seg_b;
INT16           angle;
UINT16          start_b_index,end_b_index;
double          b_val_1,b_val_2;
UINT8           enter_flag;
MIN_MAX_STRUC   min_max_info;
double          temp_dbl,tsx,tsy,td;

UINT16          tx1,ty1,tx2,ty2;

  tx1 = seg_ptr->x_1;
  ty1 = seg_ptr->y_1;
  tx2 = seg_ptr->x_2;
  ty2 = seg_ptr->y_2;
  tsx = tx2 - tx1;
  tsy = ty2 - ty1;
/* tsx and tsy will either be zero , less than , or
greater than  */

  td = seg_ptr->normal_info ;


  if ((tsx > 0) && (tsy == 0))
     seg_ptr->normal_info = 4;
  else
     td=1;

  if ((tsx > 0) && (tsy < 0))
     seg_ptr->normal_info = 5 ;
  else
     td=1;

  if ((tsx == 0) && (tsy < 0))
     seg_ptr->normal_info = 1;
```

```
        else
           td=1;

        if ((tsx < 0) && (tsy < 0))
           seg_ptr->normal_info = 9;
        else
           td=1;

        if ((tsx < 0) && (tsy == 0))
           seg_ptr->normal_info = 8;
        else
           td=1;

        if ((tsx < 0) && (tsy > 0))
           seg_ptr->normal_info = 10;
        else
           td=1;

        if ((tsx == 0) && (tsy > 0))
           seg_ptr->normal_info = 2;
        else
           td=1;

        if ((tsx > 0) && (tsy > 0))
           seg_ptr->normal_info = 6;
        else
           td=1;

      enter_flag = seg_ptr->normal_info;
      if (!swap_xy) {
        x1 = seg_ptr->x_1;
        y1 = seg_ptr->y_1;
        x2 = seg_ptr->x_2;
        y2 = seg_ptr->y_2;
        }
      else {
        x1 = seg_ptr->y_1;
```

```
      y1 = seg_ptr->x_1;
      x2 = seg_ptr->y_2;
      y2 = seg_ptr->x_2;
      exchange_xy_normal_info(&enter_flag);
      };

    b_val_1 = compute_b_value(x1,y1);
    b_val_2 = compute_b_value(x2,y2);
    if (b_val_1 < b_val_2) {
      start_b_index = ceil
((b_val_1-hatch_bottom)/hatch_b_spacing);
      end_b_index   =
floor((b_val_2-hatch_bottom)/hatch_b_spacing);
      }
    else {
      start_b_index = ceil
((b_val_2-hatch_bottom)/hatch_b_spacing);
      end_b_index   =
floor((b_val_1-hatch_bottom)/hatch_b_spacing);
      };
    if (end_b_index > MAX_POINTER_INDEX) end_b_index = 0;

    compute_line(x1,y1,x2,y2,&seg_m,&seg_b);
    if (x1<x2) {
      min_max_info.min_x = x1;
      min_max_info.max_x = x2;
      }
    else {
      min_max_info.min_x = x2;
      min_max_info.max_x = x1;
      };
    if (y1<y2) {
      min_max_info.min_y = y1;
      min_max_info.max_y = y2;
      }
    else {
      min_max_info.min_y = y2;
```

```
    min_max_info.max_y = y1;
    };

  angle = fabs((hatch_angle - atan(seg_m))/
RADIANS_PER_DEGREE);
  if (angle > 90) angle = 180-angle;

  if (seg_m == 0.0) {
    enter_flag &= NORMAL_Y_POSITIVE;
    if (hatch_m < 0.0) enter_flag ^= NORMAL_Y_POSITIVE;
    }
  else {
    enter_flag &= NORMAL_X_POSITIVE;
    temp_dbl = fabs(hatch_m); /* For Microport Bug */
/*    if ((sign(hatch_m) == sign(seg_m)) && (temp_dbl >
fabs(seg_m)))    */
/*    test to check for crosshatch flags     */

    if ((sign(hatch_m) == sign(seg_m)) && (temp_dbl >
fabs(seg_m)))
      enter_flag ^= NORMAL_X_POSITIVE;
    };
  if (enter_flag) enter_flag = OUT_FLAG;
  else enter_flag = IN_FLAG;

  while (start_b_index <= end_b_index) {
    hatch_b_value = start_b_index * hatch_b_spacing +
hatch_bottom;
    if (((fabs(seg_m-hatch_m) > ZERO_DEF) ||
         (fabs(seg_b-hatch_b_value) > ZERO_DEF)) &&

compute_intersect(seg_m,seg_b,&min_max_info,&hatch_x,&ha
tch_y))

add_point_to_table(hatch_x,hatch_y,angle,enter_flag);
    start_b_index++;
    };
```

```
  };


/*******************************************************

    get_hatch_pattern

    Takes border list and generates hatch segments


*******************************************************/

static void
get_hatch_pattern(border_seg_list,hatch_list,spacing,mia
,rot_angle)
  OUT_SEG_STRUC *border_seg_list;
  OUT_SEG_STRUC **hatch_list;
  UINT16        spacing;
  UINT16        mia;
  UINT16        rot_angle;
  {
  UINT16        i,temp_uint;
  int           loop_start,loop_end,pointer,direction;
  BOOLEAN       hatch_found,swap_xy,reverse;
  OUT_SEG_STRUC *hatch_ptr;

  if (rot_angle == 90) {
    swap_xy = TRUE;
    rot_angle = 0;
    }
  else swap_xy = FALSE;

  hatch_angle = (double) rot_angle * RADIANS_PER_DEGREE;

  hatch_m = tan(hatch_angle);
  hatch_b_spacing = spacing/cos(hatch_angle);
```

```
  if (hatch_b_spacing < 0) hatch_b_spacing =
-hatch_b_spacing;

  get_hatch_bottom(border_seg_list,swap_xy);
  hatch_max_index = 0;
  while (border_seg_list != NULL_SEG) {
    get_hatch_points(border_seg_list,swap_xy);
    border_seg_list = border_seg_list-> next_seg;
    };

  reverse = FALSE;
  loop_start = 0;
  loop_end = hatch_max_index;
  pointer = loop_start;
  direction = 1;
  do {
    do {
      get_segment(pointer,hatch_list,mia,&hatch_found);
      if (hatch_found) {
        hatch_ptr = *hatch_list;
        if (swap_xy) {
          temp_uint = hatch_ptr->x_1;
          hatch_ptr->x_1 = hatch_ptr->y_1;
          hatch_ptr->y_1 = temp_uint;
          temp_uint = hatch_ptr->x_2;
          hatch_ptr->x_2 = hatch_ptr->y_2;
          hatch_ptr->y_2 = temp_uint;
          };
        if (reverse) {
          temp_uint = hatch_ptr->x_1;
          hatch_ptr->x_1 = hatch_ptr->x_2;
          hatch_ptr->x_2 = temp_uint;
          temp_uint = hatch_ptr->y_1;
          hatch_ptr->y_1 = hatch_ptr->y_2;
          hatch_ptr->y_2 = temp_uint;
          };
        reverse = !reverse;
```

```
        );
      pointer += direction;
      } while ((pointer >= loop_start) && (pointer <=
loop_end));

    while ((hatch_point_table[loop_start] ==
NULL_HATCH_POINT) &&
           (loop_start != loop_end))
      loop_start++;
    while ((hatch_point_table[loop_end] ==
NULL_HATCH_POINT) &&
           (loop_end != loop_start))
      loop_end--;

    if (pointer < loop_start) {
      pointer = loop_start;
      direction = 1;
      }
    else {
      pointer = loop_end;
      direction = -1;
      );
  } while (hatch_point_table[loop_start] !=
NULL_HATCH_POINT);


  );


/*****************************************************

   exported procedure

   pfp_get_hatch_lines

*****************************************************/
```

```
void pfp_get_hatch_lines (border_seg_list, hatch_list,
                          horiz_spacing, vert_spacing,
                          angle_spacing,
min_intersect_angle,elim_flat_tri_flag)
  OUT_SEG_STRUC *border_seg_list;
  OUT_SEG_STRUC **hatch_list;
  UINT16        horiz_spacing;
  UINT16        vert_spacing;
  UINT16        angle_spacing;
  UINT16        min_intersect_angle;
  BOOLEAN       elim_flat_tri_flag;
  {
  UINT16        i;

  if (border_seg_list != NULL_SEG) {

    elim_flat_tri = elim_flat_tri_flag;

    if (!hatch_clear) {
      for (i=0; i < MAX_POINTER_INDEX;i++)
hatch_point_table[i] = NULL_HATCH_POINT;
      hatch_clear = TRUE;
      );

    if (horiz_spacing) {
      get_hatch_pattern (border_seg_list, hatch_list,
                         horiz_spacing,
min_intersect_angle,0);
      }
    if (vert_spacing) {
      get_hatch_pattern (border_seg_list, hatch_list,
                         vert_spacing,
min_intersect_angle,90);
      }
    if (angle_spacing) {
      get_hatch_pattern (border_seg_list, hatch_list,
```

```
                              angle_spacing,
min_intersect_angle,60);
      get_hatch_pattern (border_seg_list, hatch_list,
                              angle_spacing,
min_intersect_angle,120);
      }
    }
  } /* pfp_get_hatch_lines */
```

```
/*****************************************************/
/*                3D Systems Slice Program           */
/*          Copyright (C) 1988 3D Systems Inc.       */
/*                                                   */
/* PFPCURE.C                                         */
/*                                                   */
/* PFP Cure Compensation Routines                    */
/*****************************************************/

#include <stdio.h>
#include <math.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfpseg.h"
#include "pfpchl.h"
#include "pfputl.h"
#include "pfpcure.h"

#define MAX_INTERSECTIONS  100

static OUT_SEG_STRUC   *seg_list;
static CURE_HASH_STRUC *hash_table [MAX_HASH_INDEX];
static UINT16          last_hash_index;
static UINT16          hash_index;
static OUT_SEG_STRUC   *seg_stack;
static UINT16          intersection_x
[MAX_INTERSECTIONS];
static UINT16          intersection_y
[MAX_INTERSECTIONS];
static INT16           intersection_ptr;
static BOOLEAN         cure_initialized = FALSE;
static BOOLEAN         warning_printed;
static UINT16          cure_amount;
static UINT16          curr_layer;

extern FILE            *message_file;
```

```
extern BOOLEAN          read_message_file;
extern float            resolution;

/*******************************************************

    concat_segments

    Combines 2 segments into on if they are connected
and straight

*******************************************************/

BOOLEAN concat_segments(seg1,seg2,user_info,resolution,
cure_value)
  UINT16                  cure_value;
  USER_INFO_STRUC  *user_info;
  OUT_SEG_STRUC *seg1,*seg2;
  float resolution;
  {
  double dx1,dx2,dy1,dy2,m1,m2;
  double x1,y1,x2,y2,x3,y3;

  double mag1,mag2,magt,dot,ang,qout;

  double ang1,ang2;
  float res1;

  /* If new segment is Null then act like you combined
them */
  if (seg2==NULL_SEG) return(TRUE);
  res1 = resolution;

  if (res1 <= 1000){
  ang1 = 30;
  ang2 = 155;
  resolution = resolution * .013;
  } else {
```

```
ang1 = 30;
ang2 = 110;
resolution = resolution * .013;
}
if (cure_value == 0){
ang1 = 1;
ang2 = 179;
} else {
ang1 = ang1;
ang2 = ang2;
}


/* See if attached */
if ((seg1==NULL_SEG) ||
    (seg1->x_2!=seg2->x_1) ||
    (seg1->y_2!=seg2->y_1)) return(FALSE);
else {
  /* Attached. See if straight. */
  /* Compute Slope of segments */
  x1 = seg1->x_1;
  y1 = seg1->y_1;
  x2 = seg1->x_2;
  y2 = seg1->y_2;
  x3 = seg2->x_2;
  y3 = seg2->y_2;
  dy1 = y2 - y1;
  dy2 = y3 - y2;
  dx1 = x2 - x1;
  dx2 = x3 - x2;

/*  now we calculate the angle between the two vectors
using the dot product  */

    mag1 = sqrt((dx1*dx1)+(dy1*dy1));
    mag2 = sqrt((dx2*dx2)+(dy2*dy2));
    magt = mag1*mag2;
```

```
        dot = (dx1*dx2)+(dy1*dy2);
        qout = ( dot/magt);
        if ( qout > 1 ){
        qout = 1;
        } else {
        qout = qout ;
        }
        if ( qout < -1 ){
        qout = -1;
        } else {
        qout = qout ;
        }

        ang = acos(qout)/(RADIANS_PER_DEGREE);


        if (dx1==0) {
          m1 = 100000;
          if (dy1<0) m1= -m1;
          }
        else m1 = dy1/dx1;

        if (dx2==0) {
          m2 = 100000;
          if (dy2<0) m2= -m2;
          }
        else m2 = dy2/dx2;

/*      if  ((fabs(m1-m2)<ZERO_DEF) &&
             (sign(dx1)==sign(dx2)) &&
             (sign(dy1)==sign(dy2))) {
          seg1->x_2 = seg2->x_2;
          seg1->y_2 = seg2->y_2;
          return(TRUE);
          }                                          */

/* this is the new check for small inline segments  */
```

```
/*     if  ((fabs(fabs(m1) - fabs(m2))< 2.8  ) &&
          (sign(dx1)==sign(dx2)) &&
          (sign(dy1)==sign(dy2)) &&                */
       if  ((( ang < ang1 ) || ( ang > ang2 ))    &&

        ( (fabs(dx1)< resolution )  &&
          (fabs(dy1)< resolution )  ||
          (fabs(dx2)< resolution )  &&
          (fabs(dy2)< resolution )     )
    ) {
       seg1->x_2 = seg2->x_2;
       seg1->y_2 = seg2->y_2;
       return(TRUE);
       }
     else return(FALSE);
     );
  };
/*******************************************************

     hash_empty

     checks for hash empty and updates last index



*******************************************************/

BOOLEAN hash_empty()
  {
  while ((last_hash_index<MAX_HASH_INDEX) &&
         (hash_table[last_hash_index]==NULL_CURE_HASH))
last_hash_index++;
  return(last_hash_index==MAX_HASH_INDEX);
  );
```

```
/*********************************************************

    get_hash_index

    computes hash index

*********************************************************/

void get_hash_index(x,y)
  UINT16 x,y;
  {
  double xd,yd;
  double hash_key;

  xd=x;
  yd=y;
  hash_key = (double) (MAX_Y_VAL) * xd + yd;
  hash_index = (UINT16) fmod(hash_key,(double)
MAX_HASH_INDEX);
  );


/*********************************************************

    normal_on_left

    Tells if Normal of segment is on Left side of
segment looking from point 1 to 2.

*********************************************************/


BOOLEAN normal_on_left(seg_ptr)
  OUT_SEG_STRUC *seg_ptr;
  {
```

```
  double          a1,a2;
  double          b1,b2;
  double          i,j,k;

  /* See if point on left. Use Cross product. if "Z"
component pos then it is on the left side, otherwise on
right */
  a1 = (double) (seg_ptr->x_2) - seg_ptr->x_1;
  a2 = (double) (seg_ptr->y_2) - seg_ptr->y_1;
  b1 = a1;
  b2 = a2;
  if ((seg_ptr->normal_info&NORMAL_X_POSITIVE)!=0) b1 +=
0.01;
  else if ((seg_ptr->normal_info&NORMAL_X_NEGATIVE)!=0)
b1 -= 0.01;
  if ((seg_ptr->normal_info&NORMAL_Y_POSITIVE)!=0) b2 +=
0.01;
  else if ((seg_ptr->normal_info&NORMAL_Y_NEGATIVE)!=0)
b2 -= 0.01;
  cross_product(a1,a2,(double) 0,b1,b2,(double)
0,&i,&j,&k);
  return(k>0);
  );

/*******************************************************


    load_hash_table

    Loads the cure hash table with pointers to segments


*******************************************************/
void load_hash_table()
  {
  CURE_HASH_STRUC *cure_ptr;
```

```
  while (seg_list!=NULL_SEG) {

    /* Each segment has two endpoints to add to hash */
    cure_ptr = (CURE_HASH_STRUC *)
mem_alloc(sizeof(CURE_HASH_STRUC));

    /* Make sure normal is on left side of segment */
    if (!normal_on_left(seg_list))
swap_points(seg_list);

    /* Point to segment */
    cure_ptr->seg_ptr = seg_list;

    /* Set X and Y */
    cure_ptr->x = seg_list->x_1;
    cure_ptr->y = seg_list->y_1;

    /* Add endpoint to correct hash list */
    get_hash_index(cure_ptr->x,cure_ptr->y);
    if (hash_table[hash_index]!=NULL_CURE_HASH)
      hash_table[hash_index]->last_hash = cure_ptr;
    cure_ptr->last_hash=NULL_CURE_HASH;
    cure_ptr->next_hash = hash_table[hash_index];
    hash_table[hash_index]=cure_ptr;

    /* Next segment */
    seg_list = seg_list->next_seg;
    };
  };


/*********************************************************

    delete_cure_hash_point

    Delinks hash point
```

```
*********************************************************/
void delete_cure_hash_point(hash_ptr)
  CURE_HASH_STRUC *hash_ptr;
  {
  CURE_HASH_STRUC *last_ptr,*next_ptr;

  last_ptr=hash_ptr->last_hash;
  next_ptr=hash_ptr->next_hash;

  if (last_ptr==NULL_CURE_HASH) {
    get_hash_index(hash_ptr->x,hash_ptr->y);
    hash_table[hash_index]=next_ptr;
    }
  else last_ptr->next_hash = next_ptr;
  if (next_ptr!=NULL_CURE_HASH) next_ptr->last_hash =
last_ptr;
  mem_free(hash_ptr,sizeof(CURE_HASH_STRUC));
  };


/*********************************************************

    find_endpoint

    Looks down hash table for an endpoint.

*********************************************************/

CURE_HASH_STRUC *get_hash_endpoint(x,y)
  UINT16 x,y;
  {
  CURE_HASH_STRUC *hash_ptr;
  BOOLEAN         found;

  get_hash_index(x,y);
```

```
  hash_ptr=hash_table[hash_index];
  found=FALSE;

  while (!found && (hash_ptr!=NULL_CURE_HASH)) {
    if ((hash_ptr->x==x) && (hash_ptr->y==y))
found=TRUE;
    else hash_ptr = hash_ptr->next_hash;
    };
  return(hash_ptr);
  };


/*********************************************************
    find_intersection

    Looks through intersection list for point
*********************************************************/

BOOLEAN find_intersection(x,y,user_info)
  UINT16 x,y;
  USER_INFO_STRUC *user_info;
  {
  INT16   i;
  i=intersection_ptr;
  while ((i>=0) && ((intersection_x[i]!=x) ||
(intersection_y[i]!=y))) i--;
  if (i<0) return(FALSE);
  if ((cure_amount>0) && (i!=0) &&
(i!=intersection_ptr)) {
    read_message_file=TRUE;
    fprintf(message_file," Layer %.3g (%u) Cure
Compensation: Warning - Intersection at (%u,%u) not in
order.\n",
      curr_layer/user_info->res_value,curr_layer,x,y);
    };
  intersection_ptr=i;
  return(TRUE);
```

```
  );


/*********************************************************

    push_intersection_stack

    Pushes top of segment stack segment as intersection

*********************************************************/

void push_intersection_stack()
  {
  intersection_ptr++;
  if (intersection_ptr==MAX_INTERSECTIONS) {
    fprintf(stderr,"\nerror - Too many border
intersections on layer to process\n");
    fprintf(stderr,"        Maximum: %d\n",(INT16)
MAX_INTERSECTIONS);
    exit(SYS_FAILURE);
    );
  intersection_x[intersection_ptr] = seg_stack->x_1;
  intersection_y[intersection_ptr] = seg_stack->y_1;
  );

/*********************************************************

    push_seg_stack

    Pushes a segment onto the segment stack. Combines
the top segment with the current segment if in line.

*********************************************************/

void push_seg_stack(hash_ptr,user_info,cure_value)
  UINT16                cure_value;
  CURE_HASH_STRUC *hash_ptr;
```

```
  USER_INFO_STRUC *user_info;
  {
  OUT_SEG_STRUC *seg_ptr;

  seg_ptr=hash_ptr->seg_ptr;

  /* Add to segment stack */
  if (!concat_segments(seg_stack,seg_ptr,user_info,
resolution,cure_value)) {
    seg_ptr->next_seg = seg_stack;
    seg_stack = seg_ptr;
    }
  else mem_free(seg_ptr,sizeof(OUT_SEG_STRUC));

  /* Delete hash entry */
  delete_cure_hash_point(hash_ptr);
  };


/*******************************************************

    pop_seg_stack

    Pops top of segment stack and puts in master segment
list


*******************************************************/

void pop_seg_stack()
  {
  OUT_SEG_STRUC   *temp_seg_ptr;

  if (seg_stack!=NULL_SEG) {
    temp_seg_ptr=seg_stack;
    seg_stack = seg_stack->next_seg;
    temp_seg_ptr->next_seg=seg_list;
```

```
    seg_list=temp_seg_ptr;
    };
  };

/*********************************************************

    cure_sort

    Takes input border, sorts it end to end

*********************************************************/

void
cure_sort(in_seg_list,user_info,layer,layer_value,tt)
  UINT8            tt;
  OUT_SEG_STRUC    **in_seg_list;
  USER_INFO_STRUC *user_info;
  UINT16           layer,layer_value;
  {
  UINT16           cure_value;
  UINT16           curr_x,curr_y;
  CURE_HASH_STRUC *found_point;
  BOOLEAN          old_intersection,an_intersection;
  OUT_SEG_STRUC    *temp_seg_ptr;
  OUT_SEG_STRUC    *end_seg_list;

  /* Initialize cure vars if first time */
  if (!cure_initialized) {
    for (last_hash_index=0;
last_hash_index<MAX_HASH_INDEX; last_hash_index++)
      hash_table[last_hash_index]=NULL_CURE_HASH;
    seg_stack = NULL_SEG;
    cure_initialized=TRUE;
    };

  cure_value = user_info->cure;
  curr_layer = layer_value;
```

```
  cure_amount = cure_value;

  /* RULE 1 - Load the hash table with segment endpoints
*/
  warning_printed=FALSE;
  seg_list=(OUT_SEG_STRUC *) *in_seg_list;
  load_hash_table();


  /* RULE 2 - If hash table empty then Done
              else begin
                Push any segment on seg stack
                Order points of segment by normal so
                  will rotate in correct direction for
                  Cure Compensation
                Push as intersection
              end
              Make second endpoint then current endpoint
  */
  last_hash_index=0;
  while (!hash_empty()) {
    if (seg_stack==NULL_SEG) {
      /* Push first point on segment stack and as
intersection */
      intersection_ptr = -1;
      push_seg_stack(hash_table[last_hash_index],user_
info,cure_value);
      push_intersection_stack();
      );

    /* RULE 3 - repeat
                  Find matching endpoint in hash table
                  If not found then ERROR
                  Push matched segment on seg stack
                until current endpoint in intersection
stack
```

```
    */
    if (!hash_empty()) {
      do {
        curr_x=seg_stack->x_2;
        curr_y=seg_stack->y_2;
        old_intersection =
find_intersection(curr_x,curr_y,user_info);
        if (!old_intersection) {
          found_point =
get_hash_endpoint(curr_x,curr_y);
          if (found_point==NULL_CURE_HASH) {
            if (!warning_printed) {
              read_message_file=TRUE;
              fprintf(message_file," Layer %.3g (%u)
Warning: Incomplete Border Segments Detected\n",
                curr_layer/user_info->res_value,
curr_layer);
              warning_printed=TRUE;
              };
            old_intersection = TRUE;
            intersection_ptr = 0;
            }
          else {
            push_seg_stack(found_point,user_info,
cure_value);
            an_intersection =
get_hash_endpoint(curr_x,curr_y)!=NULL_CURE_HASH;
            };
          };
      } while (!old_intersection && !an_intersection);

      /* RULE 4 - Got Intersection Point
                  if point in intersection stack then
begin
                    create polygon by poping segment
stack to intersection
                    pop intersection stack
```

```
                    end else begin
                      push on segment stack
                      push on intersection stack
                    end
        */
        if (old_intersection) {
          temp_seg_ptr = seg_stack;
          do {
            pop_seg_stack();
          } while ((seg_stack!=NULL_SEG) &&
                   ((seg_list->x_1!=intersection_x
[intersection_ptr]) ||
                    (seg_list->y_1!=intersection_y
[intersection_ptr])));
          intersection_ptr--;
          if (concat_segments(temp_seg_ptr,seg_list,
user_info,resolution,cure_value)) {
            temp_seg_ptr = seg_list;
            seg_list = seg_list->next_seg;
            mem_free(temp_seg_ptr,sizeof(OUT_SEG_STRUC));
            };
          }
        else push_intersection_stack();
        };
      };
    *in_seg_list = seg_list;
    };

/*******************************************************

    dot_product

    returns dot porduct of length of vector

*******************************************************/

double dot_product(vx1,vy1,vx2,vy2)
```

```
  double vx1,vy1,vx2,vy2;
  {
  return(vx1*vx2+vy1*vy2);
  };

/*********************************************************

    length_of_vector

    returns length of vector

*********************************************************/

double length_of_vector(x,y)
  double x,y;
  {
  return(sqrt(x*x+y*y));
  };

/*********************************************************
    length_of_segment

    returns length of a segment
*********************************************************/

double length_of_segment(seg_ptr)
  OUT_SEG_STRUC *seg_ptr;
  {
  double x,y;

  x = (double) seg_ptr->x_2 - (double) seg_ptr->x_1;
  y = (double) seg_ptr->y_2 - (double) seg_ptr->y_1;
  return(length_of_vector(x,y));
  };

/*********************************************************
```

```
    make_normal

    Return normal of vector (Assume normal on left of
vector)

*******************************************************/

void make_normal(vx,vy,nx,ny)
  double vx,vy;
  double *nx,*ny;
  {
  double len;

  len = length_of_vector(vx,vy);
if (len != 0) {
  *nx = -vy/len;
  *ny = vx/len;
} else {
  *nx = 0;
  *ny = 0;
}
  );



/*******************************************************

    make_unit_vector

    M  es a unit vector out of line segment

*******************************************************/

void make_unit_vector(seg_ptr,vx,vy)
  OUT_SEG_STRUC *seg_ptr;
  double *vx,*vy;
  {
```

```
  double x,y,len;

  x = (double) seg_ptr->x_2 - (double) seg_ptr->x_1;
  y = (double) seg_ptr->y_2 - (double) seg_ptr->y_1;
  len = length_of_vector(x,y);
  if (len != 0) {
  *vx = x/len;
  *vy = y/len;
} else {
  *vx = 0;
  *vy = 0;
}
  );


/*********************************************************

    cure_compensate

    Cure Compensates inpute border that has been sorted

*********************************************************/

void cure_compensate(seg_list,user_info,tt)
  UINT8           tt;
  OUT_SEG_STRUC   *seg_list;
  USER_INFO_STRUC *user_info;
  {
  /*
    While there is a segment left
      Get first segment.
      StartPoint = Fisrt Point of First Segment - For
end of loop connection
      CurrentPoint = SecondPoint
      While CurrentPoint!=StartPoint
CurrentPoint==FisrtPoint of next segment
        S - Current Segment Vector
```

```
            N - Next Segment Vector
            UV -   ( S+N )
                 -(-----)
                  (|S+N|)

            |V| =     BeamRadius
                  -----------------
                  Sqrt(      (S.N) )
                      (1 -   ------)
                      (      |S||N|)
                      (-----------)
                      (    2      )

             V = |V| UV
            CurrentPoint = CurrentPoint + V
            FirstPoint of Next segment = CurrentPoint
            CurrentPoint = SecondPoint of Next segment
          End
        */
        OUT_SEG_STRUC
      *Start_segment,*S_segment,*N_segment,temp_seg;
        UINT16          Start_x,Start_y;
        double          Start_normal_x,Start_normal_y;
        double          Start_length;
        double          S_normal_x,S_normal_y;
        double          N_normal_x,N_normal_y;
        double          Start_vector_x,Start_vector_y;
        double          S_vector_x,S_vector_y;
        double          N_vector_x,N_vector_y;
        double          SN_vector_x,SN_vector_y;
        double          S_length,N_length,SN_length,V_length;
        double          UV_vector_x,UV_vector_y;
        double          new_point_x,new_point_y;
        double          temp_dbl;
        BOOLEAN         loop_arround,EOSurface;

        cure_amount = user_info->cure;
```

```
  while (seg_list!=NULL_SEG) {
    /* Save starting point of surface */
    Start_segment = seg_list;
    Start_x = seg_list->x_1;
    Start_y = seg_list->y_1;
    Start_length = length_of_segment(Start_segment);
    /* Compute Normal of starting segment */
    make_unit_vector(Start_segment,&Start_vector_x,&
Start_vector_y);
    make_normal(Start_vector_x,Start_vector_y,&Start_
normal_x,&Start_normal_y);
    /* Set up for loop of surface.  Make first segment
look as if is the Next one we are to analize */
    N_segment = Start_segment;
    N_normal_x = Start_normal_x;
    N_normal_y = Start_normal_y;
    N_vector_x = Start_vector_x;
    N_vector_y = Start_vector_y;
    N_vector_y = Start_vector_y;
    N_length   = Start_length;

    /* Loop until surface completed or find the end of
an incomplete surface */
    loop_arround = FALSE;
    EOSurface = FALSE;
    while (!loop_arround && !EOSurface) {
      /* Make last segment the current segment */
      S_segment = N_segment;
      S_normal_x = N_normal_x;
      S_normal_y = N_normal_y;
      S_vector_x = N_vector_x;
      S_vector_y = N_vector_y;
      S_length = N_length;
      /* Get the Next segment from here */
      if ((S_segment->x_2==Start_x) &&
(S_segment->y_2==Start_y)) {
```

```
        /* The current segment is the completion of
solid surface */
        /* Make the original starting segment the Next
segment */
        loop_arround = TRUE;
        N_segment = Start_segment;
        N_normal_x = Start_normal_x;
        N_normal_y = Start_normal_y;
        N_vector_x = Start_vector_x;
        N_vector_y = Start_vector_y;
        N_length = Start_length;
        seg_list = S_segment->next_seg;
        }
      else {
        N_segment = S_segment->next_seg;
        if ((N_segment==NULL_SEG) ||
            (N_segment->x_1!=S_segment->x_2) ||
            (N_segment->y_1!=S_segment->y_2)) {
          /* The Next segment does not "attach" to
current segment */
          /* Use same normal for Next segment so we at
least compensate */
          EOSurface = TRUE;
          seg_list = N_segment;
          N_segment = &temp_seg;
          N_normal_x = S_normal_x;
          N_normal_y = S_normal_y;
          N_vector_x = S_vector_x;
          N_vector_y = S_vector_y;
          N_length = S_length;
          }
        else {
          /* Compute Normal of Next segment */
          make_unit_vector(N_segment,&N_vector_x,&
N_vector_y);
```

```
make_normal(N_vector_x,N_vector_y,&N_normal_x,&
N_normal_y);
          N_length = length_of_segment(N_segment);
          };
        };
      /* Compute V */
      /* First find S+N */
      SN_vector_x = S_normal_x + N_normal_x;
      SN_vector_y = S_normal_y + N_normal_y;
      SN_length =
length_of_vector(SN_vector_x,SN_vector_y);
      if (SN_length==0) {
        /* Normals cancel out.  Impossible case of line
doubling back on self */
        /* Just back off in line direction */
        UV_vector_x = S_normal_y;
        UV_vector_y = -S_normal_x;
        }
      else {
        /* Normalize S+N */
        UV_vector_x = -SN_vector_x/SN_length;
        UV_vector_y = -SN_vector_y/SN_length;
        };
      /* Compute the length of the offset vector */
      /* If length is 0 then probably was single line
looping back.  Just offset
           by Cure amount */
      /* S.N is of the segment vectors. Use rotated
normals */
      V_length = sqrt((1.0-(-S_normal_y * N_normal_y -
S_normal_x * N_normal_x))/2.0);
      if (V_length==0) V_length = (double) cure_amount;
      else V_length = (double) cure_amount/V_length;

      /* Check for Insignificant Feature */
      /* See if V projects over 1/2 of segment S */
```

```
/* this projection limits the cure compensation --
remove        */
      if
(dot_product(UV_vector_x,UV_vector_y,S_vector_x,
S_vector_y)!=0) {
        temp_dbl = fabs((S_length / 2 - 1) /

dot_product(UV_vector_x,UV_vector_y,S_
vector_x,S_vector_y));
/*        if (V_length>temp_dbl) V_length = temp_dbl;
*/
/*   remove 1/2 length option for LB only
*/

        if ((tt==0 )&&(V_length>temp_dbl)) {
            V_length = V_length;
        } else {
        if ((tt==1)&&(V_length>(temp_dbl))) V_length =
V_length ;
        );
       );

      /* See if V projects over 1/2 of segment N */

      if
(dot_product(UV_vector_x,UV_vector_y,N_vector_x,N_vector
_y)!=0) {
        temp_dbl = fabs((N_length / 2 - 1) /

dot_product(UV_vector_x,UV_vector_y,N_vector_x,N_vector_
y));
/*        if (V_length>temp_dbl) V_length = temp_dbl;
     */
/*   remove 1/2 length option for LB only             */

        if ((tt == 0 )&&(V_length>temp_dbl)) {
            V_length = V_length;
```

```
        } else {
        if  ((tt==1)&&(V_length>(temp_dbl))) V_length =
V_length ;
        };
       };

      /* Compute cured offset point */
      new_point_x = round_dbl(S_segment->x_2 +
UV_vector_ x*V_length);
      new_point_y = round_dbl(S_segment->y_2 +
UV_vector_ y*V_length);

      /* Update segments */
      S_segment->x_2 = new_point_x;
      S_segment->y_2 = new_point_y;
      N_segment->x_1 = new_point_x;
      N_segment->y_1 = new_point_y;
      };
    };
  };
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPIDB.C                                            */
/*                                                     */
/* PFP Input Database Module                           */
/*******************************************************/

#include <stdio.h>
#include <math.h>
#include <malloc.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"


extern INPUT_TRI_STRUC *tri_bucket_array[];
extern OUT_SEG_STRUC *FD_seg_bucket_array[];
extern OUT_SEG_STRUC *FU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFD_seg_bucket_array[];
extern UINT16        z_control_table[];
extern UINT16        all_min_z, all_max_z,
all_min_z_index, all_max_z_index;
extern FILE          *message_file;

/*******************************************************
   general procedure

   find_z_binarily - binary search of z_control_table
for layer z

*******************************************************/
```

```
UINT16 get_z_binarily(z)

   UINT16 z;

{
   UINT16 low = 0, mid, high = MAX_Z_INDEX-1;

   while (low <= high) {
      mid = (low + high) / 2;
      if (z < z_control_table [mid])
         high = mid - 1;
      else if (z > z_control_table [mid])
         low = mid + 1;
      else
         return(mid);
    }
    if (z_control_table [low] > z)
       return(low);
    else return(high);

}  /* get_z_binarily */



/*******************************************************
   local procedure

   get_offsets

*******************************************************/

static void get_offsets (min_x, min_y, min_z, max_x,
max_y, max_z,
                         res_value, x_offset, y_offset,
z_offset)
```

```
   float min_x, min_y, min_z, max_x, max_y, max_z,
res_value;
   float *x_offset, *y_offset, *z_offset;

{
   BOOLEAN res_error1,res_error2;
   float   x_range, y_range, z_range;
   float   round_flt();


   x_range = (max_x - min_x) * res_value;
   y_range = (max_y - min_y) * res_value;
   z_range = (max_z - min_z) * res_value;

   res_error1 = ((x_range > MAX_X_VAL) || (y_range >
MAX_Y_VAL) ||
                  (z_range > MAX_Z_VAL));

   res_error2 = (z_range > z_control_table
[MAX_Z_INDEX-1]);

   if (res_error1 || res_error2) {
      if (res_error1) {
         fprintf (stderr, "\n error - resolution value
too large for this");
         fprintf (stderr, " part\n");
      }
      if (res_error2) {
         fprintf (stderr, "\n error - z-value would
require more than");
         fprintf (stderr, " %u layers.\n",MAX_Z_INDEX);
      }
      fprintf (stderr,
      " min x = %f max x = %f min y = %f max y = %f min
z = %f max z = %f\n",
        min_x, max_x, min_y, max_y, min_z, max_z);
```

```
      exit (SYS_FAILURE);
   }

   *x_offset = round_flt(((MAX_X_VAL - x_range) / 2.0) -
(min_x * res_value));
   *y_offset = round_flt(((MAX_Y_VAL - y_range) / 2.0) -
(min_y * res_value));
   *z_offset = round_flt(((z_control_table
[MAX_Z_INDEX-1] - z_range) / 2.0) - (min_z *
res_value));

} /* get_offsets */



/*******************************************************
   local procedure

   check_tri_dimensions

*******************************************************/

static void check_tri_dimensions (tri_ptr)

   INPUT_TRI_STRUC *tri_ptr;

{
   if ((tri_ptr-> x1 > MAX_X_VAL) || (tri_ptr-> x1 <
0.0) ||
       (tri_ptr-> x2 > MAX_X_VAL) || (tri_ptr-> x2 <
0.0) ||
       (tri_ptr-> x3 > MAX_X_VAL) || (tri_ptr-> x3 <
0.0) ||
```

```
        (tri_ptr-> y1 > MAX_Y_VAL) || (tri_ptr-> y1 <
0.0) ||
        (tri_ptr-> y2 > MAX_Y_VAL) || (tri_ptr-> y2 <
0.0) ||
        (tri_ptr-> y3 > MAX_Y_VAL) || (tri_ptr-> y3 <
0.0) ||
        (tri_ptr-> z1 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z1 < 0.0) ||
        (tri_ptr-> z2 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z2 < 0.0) ||
        (tri_ptr-> z3 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z3 < 0.0))
      {
        fprintf (stderr, "\n error - dimensions of
uncentered object");
        fprintf (stderr, " exceed limits.  Check
Resolution value. \n");
        exit (SYS_FAILURE);
      }

} /* check_tri_dimensions */




/*********************************************************

    ocal procedure

   res_triangles

*********************************************************/
```

```
static void res_triangles (tri_list, x_offset, y_offset,
z_offset,
                                res_value, centering_off)

   INPUT_TRI_STRUC *tri_list;
   float           x_offset, y_offset, z_offset,
res_value;
   BOOLEAN         centering_off;

{
   INPUT_TRI_STRUC *tri_ptr;
   float           round_flt();


   tri_ptr = tri_list;

   if (centering_off)
      while (tri_ptr != NULL_INPUT_TRI)
         {
          tri_ptr-> x1 = round_flt(tri_ptr-> x1 *
res_value);
          tri_ptr-> x2 = round_flt(tri_ptr-> x2 *
res_value);
          tri_ptr-> x3 = round_flt(tri_ptr-> x3 *
res_value);

          tri_ptr-> y1 = round_flt(tri_ptr-> y1 *
res_value);
          tri_ptr-> y2 = round_flt(tri_ptr-> y2 *
res_value);
          tri_ptr-> y3 = round_flt(tri_ptr-> y3 *
res_value);

          tri_ptr-> z1 = round_flt(tri_ptr-> z1 *
res_value);
          tri_ptr-> z2 = round_flt(tri_ptr-> z2 *
res_value);
```

```
            tri_ptr-> z3 = round_flt(tri_ptr-> z3 *
        res_value);

            check_tri_dimensions (tri_ptr);
            tri_ptr = tri_ptr-> next_input_tri;
           }
      else
         while (tri_ptr != NULL_INPUT_TRI)
            {
            tri_ptr-> x1 = round_flt((tri_ptr-> x1 *
        res_value) + x_offset);
            tri_ptr-> x2 = round_flt((tri_ptr-> x2 *
        res_value) + x_offset);
            tri_ptr-> x3 = round_flt((tri_ptr-> x3 *
        res_value) + x_offset);

            tri_ptr-> y1 = round_flt((tri_ptr-> y1 *
        res_value) + y_offset);
            tri_ptr-> y2 = round_flt((tri_ptr-> y2 *
        res_value) + y_offset);
            tri_ptr-> y3 = round_flt((tri_ptr-> y3 *
        res_value) + y_offset);

            tri_ptr-> z1 = round_flt((tri_ptr-> z1 *
        res_value) + z_offset);
            tri_ptr-> z2 = round_flt((tri_ptr-> z2 *
        res_value) + z_offset);
            tri_ptr-> z3 = round_flt((tri_ptr-> z3 *
        res_value) + z_offset);
            tri_ptr = tri_ptr-> next_input_tri;
           }

      }  /* res_triangles */


      /*******************************************************
```

```
   local procedure

   remove_collapsed_triangles

*********************************************************


static void remove_collapsed_triangles (tri_list)

   INPUT_TRI_STRUC **tri_list;

{
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
   INPUT_TRI_STRUC head_rec;
   float           Ax,Ay,Az;
   float           Bx,By,Bz;
   float           Cx,Cy,Cz;
   unsigned long   point_count = 0,line_count = 0;
   BOOLEAN         delete_tri;


   tri_ptr = *tri_list;
   head_rec .next_input_tri = tri_ptr;
   back_ptr = &head_rec;

   while (tri_ptr != NULL_INPUT_TRI) {
     delete_tri = FALSE;
     if ((tri_ptr->x1 == tri_ptr->x2) && (tri_ptr->x2 ==
tri_ptr->x3) &&
         (tri_ptr->y1 == tri_ptr->y2) && (tri_ptr->y2 ==
tri_ptr->y3) &&
         (tri_ptr->z1 == tri_ptr->z2) && (tri_ptr->z2 ==
tri_ptr->z3)) {
       delete_tri = TRUE;
       point_count++;
       }
```

```
      else {
        Ax = tri_ptr-> x2 - tri_ptr-> x1;
        Ay = tri_ptr-> y2 - tri_ptr-> y1;
        Az = tri_ptr-> z2 - tri_ptr-> z1;
        Bx = tri_ptr-> x3 - tri_ptr-> x1;
        By = tri_ptr-> y3 - tri_ptr-> y1;
        Bz = tri_ptr-> z3 - tri_ptr-> z1;
        Cx = fabs(Ay*Bz-Az*By);
        Cy = fabs(Az*Bx-Ax*Bz);
        Cz = fabs(Ax*By-Ay*Bx);
        if ((Cx < ZERO_DEF) && (Cy < ZERO_DEF) && (Cz <
ZERO_DEF)) {
          delete_tri = TRUE;
          line_count++;
          };
        };
      if (delete_tri) {
        back_ptr-> next_input_tri = tri_ptr->
next_input_tri;
        mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
        tri_ptr = back_ptr-> next_input_tri;
        }
      else {
        tri_ptr = tri_ptr-> next_input_tri;
        back_ptr = back_ptr-> next_input_tri;
        };
       };

   *tri_list = head_rec .next_input_tri;

   if (point_count > 0) {
     fprintf (stderr, "\n warning - %lu collapsed point
triangles removed \n",
              point_count);
     fprintf (message_file, "\n warning - %lu collapsed
point triangles removed \n",
              point_count);
```

```
      );
   if (line_count > 0) {
      if (point_count == 0) {
        fprintf(stderr,"\n");
        fprintf(message_file,"\n");
        );
      fprintf (stderr, " warning - %lu collapsed line
triangles removed \n",
               line_count);
      fprintf (message_file, " warning - %lu collapsed
line triangles removed \n",
               line_count);
     );

} /* remove_collapsed_triangles */




/*********************************************************
   local procedure

   get_z_output_coord

*********************************************************/

static UINT16 get_z_output_coord (z)

   UINT16 z;

{
   UINT16 i, greater_z, lesser_z;

   i = get_z_binarily(z);
   greater_z = z_control_table [i];
```

```
   if (z == z_control_table [i])
      lesser_z = z_control_table [i];
   else
      lesser_z = z_control_table [i-1];
/* printf("get_z_output_coord:  z=%u i=%u lesser_z=%u
greater_z=%u\n",z,i,lesser_z,greater_z); */

   if ((lesser_z == 0) || (((greater_z - z) <= (z -
lesser_z)) &&
                                    (greater_z !=
z_control_table [MAX_Z_INDEX-1])))
      return (greater_z);
   else
      return (lesser_z);

} /* get_z_output_coord */



/*******************************************************
   local procedure

   adjust_z_coords

*******************************************************/


static void adjust_z_coords (tri_list, z_spacing)

   INPUT_TRI_STRUC *tri_list;
   UINT16          z_spacing;

{
   INPUT_TRI_STRUC *tri_ptr;
   float           round_flt();
   UINT16          get_z_coord();
```

```
    if (z_spacing != 1)
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = round_flt((float) tri_ptr-> z1
/ z_spacing) * z_spacing;
            tri_ptr-> z2 = round_flt((float) tri_ptr-> z2
/ z_spacing) * z_spacing;
            tri_ptr-> z3 = round_flt((float) tri_ptr-> z3
/ z_spacing) * z_spacing;
            tri_ptr = tri_ptr-> next_input_tri;
           }
       }
    else
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = get_z_output_coord ((UINT16)
tri_ptr-> z1);
            tri_ptr-> z2 = get_z_output_coord ((UINT16)
tri_ptr-> z2);
            tri_ptr-> z3 = get_z_output_coord ((UINT16)
tri_ptr-> z3);
            tri_ptr = tri_ptr-> next_input_tri;
           }
       }

} /* adjust_z_coords */
```

```
/*********************************************************

   local procedure

   output_internal_triangles

*********************************************************/


static void output_internal_triangles
(user_info,tri_list)

USER_INFO_STRUC *user_info;
INPUT_TRI_STRUC **tri_list;

{
FILE *InternalFile;
INPUT_TRI_STRUC *tri_ptr;
UINT32 count,tri_count;
float zero,i,j,k;

   zero = 0;
   tri_ptr = *tri_list;
   tri_count = 0;
   while (tri_ptr != NULL_INPUT_TRI) {
      tri_count++;
      tri_ptr = tri_ptr-> next_input_tri;
   }
   if (tri_count>0) {
      fprintf(stderr, " writing internal triangles
output file ...\n");
      InternalFile = fopen(user_info->
internal_filename,"wb");

      fprintf(InternalFile,"Slice Internal Triangle
File");
      count = 28;
```

```
      while (count<80) {              /* should write
out copy of org header */
         fprintf(InternalFile," ");
         count++;
      }
      fwrite(&tri_count,4,1,InternalFile);  /* write out
# of triangle facets */
      tri_ptr = *tri_list;
      while (tri_ptr != NULL_INPUT_TRI) {
         count = 3;
         if (tri_ptr->normal_info&NORMAL_X_POSITIVE) i =
1;
         else if
(tri_ptr->normal_info&NORMAL_X_NEGATIVE) i = -1;
              else {
                i = 0;
                count--;
                };
         if (tri_ptr->normal_info&NORMAL_Y_POSITIVE) j =
1;
         else if
(tri_ptr->normal_info&NORMAL_Y_NEGATIVE) j = -1;
              else {
                j = 0;
                count--;
                };
         if (tri_ptr->normal_info&NORMAL_Z_POSITIVE) k =
1;
         else if
(tri_ptr->normal_info&NORMAL_Z_NEGATIVE) k = -1;
              else {
                k = 0;
                count--;
                };
         switch (count) {
           case 2 : i *= 0.707106781;
                    j *= 0.707106781;
```

```
                        k *= 0.707106781;
                        break;
                case 3 : i *= 0.577350269;
                        j *= 0.577350269;
                        k *= 0.577350269;
                        break;
                };
            fwrite(&i,4,1,InternalFile);
            fwrite(&j,4,1,InternalFile);
            fwrite(&k,4,1,InternalFile);

            fwrite(tri_ptr,36,1,InternalFile);   /* write
out triangle vertices */
            fwrite(&zero,2,1,InternalFile);            /*
attribute data (none) */
            tri_ptr = tri_ptr-> next_input_tri;
        }
        fclose(InternalFile);
    }
}



/*********************************************************

    local function

    flat_tri

*********************************************************/

static flat_tri (tri_ptr)

    INPUT_TRI_STRUC *tri_ptr;
```

```
{
   return
     ((tri_ptr-> z1 == tri_ptr-> z2) && (tri_ptr-> z2 ==
tri_ptr-> z3));

} /* flat_tri */



/*********************************************************

   local function

   near_flat_tri


*********************************************************/

static near_flat_tri (tri_ptr, min_angle)

   INPUT_TRI_STRUC *tri_ptr;
   INT16           min_angle;

{
   double dx1, dy1, dz1, dx2, dy2, dz2, dx, dy, dz, d,
angle;

   if (!min_angle)
      return (FALSE);
   else
      {
       dx1 = tri_ptr-> x1 - tri_ptr-> x2;
       dy1 = tri_ptr-> y1 - tri_ptr-> y2;
```

```
        dz1 = tri_ptr-> z1 - tri_ptr-> z2;

        dx2 = tri_ptr-> x1 - tri_ptr-> x3;
        dy2 = tri_ptr-> y1 - tri_ptr-> y3;
        dz2 = tri_ptr-> z1 - tri_ptr-> z3;

        dx = dy1 * dz2 - dz1 * dy2;
        dy = dz1 * dx2 - dx1 * dz2;
        dz = dx1 * dy2 - dy1 * dx2;

        d = sqrt (dx * dx + dy * dy);

        if (d == 0) angle = 0;
        else angle = 90.0 - fabs (atan (dz / d) /
RADIANS_PER_DEGREE);

        return (angle < (double) min_angle);
       }

} /* near_flat_tri */

/*******************************************************


  cross_product

  Computes the cross product of 2 vectors


*******************************************************/
void cross_product(a1,a2,a3,b1,b2,b3,i,j,k)
  double a1,a2,a3;  /* Point A */
  double b1,b2,b3;  /* Point B */
  double *i,*j,*k;  /* A X B */
  {
  /*
```

```
    Normal i,j,k - AxB = (a2*b3 - a3*b2)i + (a3*b1 -
a1*b3)j + (a1*b2 - a2*b1)k
  */

  *i = a2*b3 - a3*b2;
  *j = a3*b1 - a1*b3;
  *k = a1*b2 - a2*b1;
  );


/*********************************************************

   recompute_normals

   Recomputes the normals of the triangles after they
have been deformed by
    adjusting the Z coords.

   Version 252 - Only recompute X and Y

*********************************************************/
static void recompute_normals(tri_list,axis)
  INPUT_TRI_STRUC *tri_list;
  INT16           axis;
  {
  UINT8  flag;
  double  a1,a2,a3;
  double  b1,b2,b3;
  double  i,j,k;
  double  temp;

  /*
    Vector A - Vertex 2 to Vertex 3 of input
    Vector B - Vertex 2 to Vertex 1 of input

    Note: Vectors change if axis is not inputs (-x -y).
Must adjust for
```

```
            these cases.
          */

          while (tri_list != NULL_INPUT_TRI) {
            switch (axis) {
              case 0 : a1 = tri_list->x3 - tri_list->x2; /* No
        axis swap */
                       a2 = tri_list->y3 - tri_list->y2;
                       a3 = tri_list->z3 - tri_list->z2;
                       b1 = tri_list->x1 - tri_list->x2;
                       b2 = tri_list->y1 - tri_list->y2;
                       b3 = tri_list->z1 - tri_list->z2;
                       break;
              case 1 : a3 = tri_list->x3 - tri_list->x2; /* X
        and Z axis swapped */
                       a2 = tri_list->y3 - tri_list->y2;
                       a1 = tri_list->z3 - tri_list->z2;
                       b3 = tri_list->x1 - tri_list->x2;
                       b2 = tri_list->y1 - tri_list->y2;
                       b1 = tri_list->z1 - tri_list->z2;
                       break;
              case 2 : a1 = tri_list->x3 - tri_list->x2; /* Y
        and Z axis swapped */
                       a3 = tri_list->y3 - tri_list->y2;
                       a2 = tri_list->z3 - tri_list->z2;
                       b1 = tri_list->x1 - tri_list->x2;
                       b3 = tri_list->y1 - tri_list->y2;
                       b2 = tri_list->z1 - tri_list->z2;
                       break;
              };

            cross_product(a1,a2,a3,b1,b2,b3,&i,&j,&k);

            switch (axis) {                          /* reswap
        flag info */
              case 1: temp = k;
                      k = i;
```

```
                i = temp;
                break;
        case 2: temp = k;
                k = j;
                j = temp;
                break;
        };

    flag = tri_list->normal_info &
(NORMAL_Z_POSITIVE+NORMAL_Z_NEGATIVE);

    if (fabs(i) > ZERO_DEF) {
      if (i < 0) flag |= NORMAL_X_NEGATIVE;
      else flag |= NORMAL_X_POSITIVE;
      };

    if (fabs(j) > ZERO_DEF) {
      if (j < 0) flag |= NORMAL_Y_NEGATIVE;
      else flag |= NORMAL_Y_POSITIVE;
      };

/*
    if (fabs(k) > ZERO_DEF) {
      if (k < 0) flag |= NORMAL_Z_NEGATIVE;
      else flag |= NORMAL_Z_POSITIVE;
      };
*/


    tri_list->normal_info = flag;
    tri_list = tri_list->next_input_tri;
    };
  };

/*******************************************************

   local procedure
```

```
*****************************************************/

static void get_tri_lists (in_file, user_info,
res_value, min_angle,
                           centering_off, z_spacing,
                           scan_tri_list, flat_tri_list,
near_flat_tri_list)

   FILE             *in_file;
   USER_INFO_STRUC *user_info;
   float            res_value;
   INT16            min_angle;
   BOOLEAN          centering_off;
   UINT16           z_spacing;
   INPUT_TRI_STRUC **scan_tri_list;
   INPUT_TRI_STRUC **flat_tri_list;
   INPUT_TRI_STRUC **near_flat_tri_list;

{
   INPUT_TRI_STRUC *input_tri_list, *tri_ptr, *temp_ptr;

   unsigned long   up_scan_tri_count = 0,
dn_scan_tri_count = 0;
   unsigned long   up_flat_tri_count = 0,
dn_flat_tri_count = 0;
   unsigned long   up_near_flat_tri_count = 0,
dn_near_flat_tri_count = 0;
   float           min_x, min_y, min_z, max_x, max_y,
max_z;
   float           x_offset, y_offset, z_offset;
   long            resoluted_min_z,resoluted_max_z;


   min_x = MAX_X_VAL;
   min_y = MAX_Y_VAL;
   min_z = z_control_table [MAX_Z_INDEX-1];
```

```
   max_x = 0.0 - MAX_X_VAL;
   max_y = 0.0 - MAX_Y_VAL;
   max_z = 0.0 - z_control_table [MAX_Z_INDEX-1];

/*
printf ("\n get_tri_lists - begin \n");
*/
   fprintf (stderr, "\n loading input database ...\n");

   if (user_info-> binary_input)
      pfp_get_input_tri_list_bin (in_file,
&input_tri_list, &min_x, &min_y,
                                   &min_z, &max_x,
&max_y, &max_z, user_info);
   else
      pfp_get_input_tri_list (in_file, &input_tri_list,
&min_x, &min_y,
                                &min_z, &max_x, &max_y,
&max_z,user_info);

   resoluted_min_z = round_flt(min_z * res_value);
   resoluted_max_z = round_flt(max_z * res_value);
   if (resoluted_min_z < z_control_table[0]) {
     if (*(user_info-> z_space_filename) != '\0')
       fprintf(stderr,"\nError - Part begins below
Variable Z Spacing Table\n");
     else
       fprintf(stderr,"\nError - Part begins below
minimum Z value\n");
     fprintf(stderr," Z Spacing Table: %u Part:
%ld\n",z_control_table[0],resoluted_min_z);
     exit (SYS_FAILURE);
     );
   if (resoluted_max_z > z_control_table[MAX_Z_INDEX-1])
{
     if (*(user_info-> z_space_filename) != '\0')
```

```
        fprintf(stderr,"\nError - Part extends past the
Variable Z Spacing Table of %u
layers\n",(MAX_Z_INDEX-1));
      else
        fprintf(stderr,"\nError - Part extends past the
maximum Z value of %u layers\n",(MAX_Z_INDEX-1));
      fprintf(stderr," Z Spacing Table: %u Part:
%u\n",z_control_table[MAX_Z_INDEX-1],resoluted_max_z);
      exit (SYS_FAILURE);
      };

   get_offsets (min_x, min_y, min_z, max_x, max_y,
max_z, res_value,
                &x_offset, &y_offset, &z_offset);

   res_triangles (input_tri_list, x_offset, y_offset,
z_offset,
                  res_value, centering_off);

   adjust_z_coords (input_tri_list, z_spacing);

   remove_collapsed_triangles (&input_tri_list);

   recompute_normals(input_tri_list,user_info->axis);

   if (*(user_info-> internal_filename) > '\0')
      output_internal_triangles
(user_info,&input_tri_list);

   while (input_tri_list != NULL_INPUT_TRI)
      {
       tri_ptr = input_tri_list;
       input_tri_list = input_tri_list-> next_input_tri;


       if (flat_tri (tri_ptr))
          {
```

```
            tri_ptr->normal_info = (tri_ptr->normal_info

&(255-NORMAL_X_NEGATIVE-NORMAL_X_POSITIVE

-NORMAL_Y_NEGATIVE-NORMAL_Y_POSITIVE));
            tri_ptr-> next_input_tri = *flat_tri_list;
            *flat_tri_list = tri_ptr;
            if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
               up_flat_tri_count++;
            else
               dn_flat_tri_count++;
           }
        else if (near_flat_tri (tri_ptr, min_angle))
           {
            temp_ptr = (INPUT_TRI_STRUC *) mem_alloc
(sizeof (INPUT_TRI_STRUC));
            copy (tri_ptr, temp_ptr, sizeof
(INPUT_TRI_STRUC));
            temp_ptr-> next_input_tri = *scan_tri_list;
            *scan_tri_list = temp_ptr;
            tri_ptr-> next_input_tri =
*near_flat_tri_list;
            *near_flat_tri_list = tri_ptr;
            if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
               up_near_flat_tri_count++;
            else
               dn_near_flat_tri_count++;
           }
        else
           {
            tri_ptr-> next_input_tri = *scan_tri_list;
            *scan_tri_list = tri_ptr;
            if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
               up_scan_tri_count++;
```

```
            else
               dn_scan_tri_count++;
            }
         }

   fprintf (stderr, "    %5lu flat triangles loaded
(%5lu up facing, %5lu down facing)\n",
                        up_flat_tri_count +
dn_flat_tri_count,
                        up_flat_tri_count,
dn_flat_tri_count);
   fprintf (stderr, "    %5lu near flat triangles loaded
(%5lu up facing, %5lu down facing)\n",
                        up_near_flat_tri_count +
dn_near_flat_tri_count,
                        up_near_flat_tri_count,
dn_near_flat_tri_count);
   fprintf (stderr, "    %5lu steep triangles loaded
(%5lu up facing, %5lu down facing)\n\n",
                        up_scan_tri_count +
dn_scan_tri_count,
                        up_scan_tri_count,
dn_scan_tri_count);



   fprintf (message_file, " %5lu flat triangles loaded
  (%5lu up facing, %5lu down facing)\n",
                        up_flat_tri_count +
dn_flat_tri_count,
                        up_flat_tri_count,
dn_flat_tri_count);
   fprintf (message_file, " %5lu near flat triangles
loaded (%5lu up facing, %5lu down facing)\n",
                        up_near_flat_tri_count +
dn_near_flat_tri_count,
```

```
                        up_near_flat_tri_count,
dn_near_flat_tri_count);
   fprintf (message_file, " %5lu steep triangles loaded
  (%5lu up facing, %5lu down facing)\n\n",
                        up_scan_tri_count +
dn_scan_tri_count,
                        up_scan_tri_count,
dn_scan_tri_count);

/*
printf ("\n get_tri_lists - end \n");
*/

} /* get_tri_lists */



/*******************************************************

   local procedure

   insert_tri

*******************************************************/

static void insert_tri (input_tri, z_bucket_array)

   INPUT_TRI_STRUC *input_tri;
   INPUT_TRI_STRUC **z_bucket_array;

{
   float          min_z;
   UINT16         min_z_value;
   UINT16         i;

   pfp_get_min_z (input_tri, &min_z);
   min_z_value = (UINT16) min_z;
```

```
    i = get_z_binarily(min_z_value);

    input_tri-> next_input_tri = z_bucket_array [i];
    z_bucket_array [i] = input_tri;

} /* insert_tri */



void tri_bucket_sort (tri_list, a_tri_bucket_array)

    INPUT_TRI_STRUC **tri_list;
    INPUT_TRI_STRUC **a_tri_bucket_array;

{
    INPUT_TRI_STRUC *tri_ptr, **bucket_ptr;
    INT16           i;

/*
printf ("\n tri_bucket_sort - begin \n");
*/
    bucket_ptr = a_tri_bucket_array;

    for (i = 0; i < MAX_Z_INDEX; i++, bucket_ptr++)
       *bucket_ptr = NULL_INPUT_TRI;

    while (*tri_list != NULL_INPUT_TRI)
       {
        t  i_ptr = (*tri_list)-> next_input_tri;
        i  sert_tri (*tri_list, a_tri_bucket_array);
        *tri_list = tri_ptr;
       }

    *tri_list = NULL_INPUT_TRI;
```

```
/*
printf ("\n tri_bucket_sort - end \n");
*/
} /* tri_bucket_sort */



void pfp_get_input_database (in_file, user_info)

   FILE            *in_file;
   USER_INFO_STRUC *user_info;

{
   INPUT_TRI_STRUC *scan_tri_list;
   INPUT_TRI_STRUC *flat_tri_list;
   INPUT_TRI_STRUC *near_flat_tri_list;
   UINT16          z;


   scan_tri_list      = NULL_INPUT_TRI;
   flat_tri_list      = NULL_INPUT_TRI;
   near_flat_tri_list = NULL_INPUT_TRI;

   get_tri_lists (in_file, user_info,
                  user_info-> res_value, user_info->
min_surface_angle,
                  user_info-> centering_off, user_info->
z_spacing,
                  &scan_tri_list, &flat_tri_list,
&near_flat_tri_list);

  for (z = 0; z < MAX_Z_INDEX; z++)
    {
     FU_seg_bucket_array[z] = NULL_SEG;
     FD_seg_bucket_array[z] = NULL_SEG;
     NFU_seg_bucket_array[z] = NULL_SEG;
```

```
      NFD_seg_bucket_array[z] = NULL_SEG;
      }

   if (scan_tri_list != NULL_INPUT_TRI)
      tri_bucket_sort (&scan_tri_list,
tri_bucket_array);

   if (flat_tri_list != NULL_INPUT_TRI)
      process_flat_tri_list (flat_tri_list);

   if (near_flat_tri_list != NULL_INPUT_TRI)
      process_near_flat_tri_list (near_flat_tri_list);

} /* pfp_get_input_database */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPTRI.C                                          */
/*                                                   */
/* PFP Cross-Hatch Line Header                       */
/*****************************************************/


#include <stdio.h>
#include <math.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"


extern INPUT_TRI_STRUC *tri_bucket_array[];
extern OUT_SEG_STRUC *FD_seg_bucket_array[];
extern OUT_SEG_STRUC *FU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFD_seg_bucket_array[];
extern UINT16        z_control_table[];

#ifdef mips
#define DISPLAY_RATE 100
#else
#define DISPLAY_RATE 10
#endif

/*****************************************************

    local procedure
```

```
    add_near_flat_segment

    Adds a segment list


*****************************************************/
static void add_near_flat_segment(segment,zi)
  OUT_SEG_STRUC **segment;
  UINT16        zi;
  {
  OUT_SEG_STRUC *seg_ptr;

  seg_ptr = *segment;
  *segment = NULL_SEG;

  if (seg_ptr->normal_info&NORMAL_Z_POSITIVE) {
    seg_ptr->next_seg = NFU_seg_bucket_array[zi];
    NFU_seg_bucket_array[zi] = seg_ptr;
    }
  else {
    seg_ptr->next_seg = NFD_seg_bucket_array[zi];
    NFD_seg_bucket_array[zi] = seg_ptr;
    };
  };


/*******************************************************

    procedure

    process_flat_tri_list

    Takes each flat triangle and generates segments for
z level
     and adds to FU or FD segment lists
```

```
*****************************************************/
void process_flat_tri_list(flat_tri_list)
  INPUT_TRI_STRUC *flat_tri_list;
  {
  OUT_SEG_STRUC   *seg_list;
  INPUT_TRI_STRUC *temp_tri_ptr;
  UINT16          flat_tri_count = 0;
  UINT16          zi;

  while (flat_tri_list != NULL_INPUT_TRI) {
    flat_tri_count++;
    if (flat_tri_count%DISPLAY_RATE == 0)
      fprintf(stderr," processing flat triangles
(%u)\r",flat_tri_count);

    seg_list = NULL_SEG;
    pfp_convert_tri_to_segs(flat_tri_list,&seg_list);
    zi = get_z_binarily((UINT16) flat_tri_list->z1);

    if (flat_tri_list->normal_info&NORMAL_Z_POSITIVE) {

append_seg_list(&seg_list,&FU_seg_bucket_array[zi]);
      FU_seg_bucket_array[zi] = seg_list;
      }
    else {

append_seg_list(&seg_list,&FD_seg_bucket_array[zi]);
      FD_seg_bucket_array[zi] = seg_list;
      );

    temp_tri_ptr = flat_tri_list;
    flat_tri_list = flat_tri_list->next_input_tri;
    mem_free(temp_tri_ptr,sizeof(INPUT_TRI_STRUC));
    );
    fprintf(stderr," processing flat triangles
(%u)\n",flat_tri_count);
```

```
};


/*******************************************************

    procedure

    process_near_flat_tri_list

    Takes each near flat triangle and breaks into
polygon for each z level
     and adds to NFU or NFD segment lists

*******************************************************/
void process_near_flat_tri_list(near_flat_tri_list)
  INPUT_TRI_STRUC *near_flat_tri_list;
  {
  OUT_SEG_STRUC
*side1_seg,*side2_seg,*top_seg,*bot_seg;
  INPUT_TRI_STRUC *temp_tri_ptr;
  VECT_TRI_STRUC  vect_tri;
  UINT16          near_tri_count = 0;
  UINT16          zi;
  float           temp;

  top_seg = NULL_SEG;
  bot_seg = NULL_SEG;

  while (near_flat_tri_list != NULL_INPUT_TRI) {
    near_tri_count++;
    if (near_tri_count%DISPLAY_RATE == 0)
      fprintf(stderr," processing near flat triangles
(%u)\r",near_tri_count);
```

```
pfp_convert_tri_to_vect(near_flat_tri_list,&vect_tri);
    zi = get_z_binarily((UINT16) vect_tri.vert1_z);

    temp = vect_tri.vert3_z;
 /*   while (z_control_table[zi+1] <= vect_tri.vert3_z)
( */
    while (z_control_table[zi+1] <= vect_tri.vert3_z) {
    temp = vect_tri.vert3_z;

pfp_get_intersection_segment(&vect_tri,z_control_table[z
i],  &bot_seg);

/*
pfp_get_intersection_segment(&vect_tri,z_control_table[z
i+1],&top_seg); */
/*   correct for laser cure compensation by moving inner
z seg interset up   */
/*   2 layers
                   */


pfp_get_intersection_segment(&vect_tri,z_control_table[z
i+1],&top_seg);

      side1_seg = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
      side1_seg->x_1 = bot_seg->x_1;
      side1_seg->y_1 = bot_seg->y_1;
      side1_seg->x_2 = top_seg->x_1;
      side1_seg->y_2 = top_seg->y_1;
      side2_seg = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
      side2_seg->x_1 = bot_seg->x_2;
      side2_seg->y_1 = bot_seg->y_2;
      side2_seg->x_2 = top_seg->x_2;
      side2_seg->y_2 = top_seg->y_2;
```

```
      side1_seg->normal_info = bot_seg->normal_info;
      side2_seg->normal_info = bot_seg->normal_info;
      /* Check for point segments in Top and Bottom. If
so, at a vertex */
      if ((bot_seg->x_1 == bot_seg->x_2) &&
(bot_seg->y_1 == bot_seg->y_2)) {
        pfp_get_xy_normal_info((float)
side1_seg->x_1,(float) side1_seg->y_1,
                               (float)
side1_seg->x_2,(float) side1_seg->y_2,
                               (float) top_seg->x_2,
(float) top_seg->y_2,
                               &side1_seg->normal_info);

        pfp_get_xy_normal_info((float)
side2_seg->x_1,(float) side2_seg->y_1,
                               (float)
side2_seg->x_2,(float) side2_seg->y_2,
                               (float) top_seg->x_1,
(float) top_seg->y_1,
                               &side2_seg->normal_info);

        pfp_get_xy_normal_info((float)
top_seg->x_1,(float) top_seg->y_1,
                               (float)
top_seg->x_2,(float) top_seg->y_2,
                               (float)
bot_seg->x_1,(float) bot_seg->y_1,
                               &top_seg->normal_info);
        add_near_flat_segment(&top_seg,zi);
        }
      else {
        pfp_get_xy_normal_info((float)
side1_seg->x_1,(float) side1_seg->y_1,
                               (float)
side1_seg->x_2,(float) side1_seg->y_2,
```

```
                                    (float) bot_seg->x_2,
(float) bot_seg->y_2,
                                    &side1_seg->normal_info);

        pfp_get_xy_normal_info((float)
side2_seg->x_1,(float) side2_seg->y_1,
                                    (float)
side2_seg->x_2,(float) side2_seg->y_2,
                                    (float) bot_seg->x_1,
(float) bot_seg->y_1,
                                    &side2_seg->normal_info);

        pfp_get_xy_normal_info((float)
bot_seg->x_1,(float) bot_seg->y_1,
                                    (float)
bot_seg->x_2,(float) bot_seg->y_2,
                                    (float)
top_seg->x_1,(float) top_seg->y_1,
                                    &bot_seg->normal_info);
        if ((top_seg->x_1 != top_seg->x_2) ||
(top_seg->y_1 != top_seg->y_2)) {
          pfp_get_xy_normal_info((float)
top_seg->x_1,(float) top_seg->y_1,
                                      (float)
top_seg->x_2,(float) top_seg->y_2,
                                      (float)
bot_seg->x_1,(float) bot_seg->y_1,
                                      &top_seg->normal_info);

          add_near_flat_segment(&top_seg,zi);
          );
        add_near_flat_segment(&bot_seg,zi);
        );
      add_near_flat_segment(&side1_seg,zi);
      add_near_flat_segment(&side2_seg,zi);
      zi++;
      );
```

```
      temp_tri_ptr = near_flat_tri_list;
      near_flat_tri_list =
near_flat_tri_list->next_input_tri;
      mem_free(temp_tri_ptr,sizeof(INPUT_TRI_STRUC));
      };

    if (top_seg != NULL_SEG)
      mem_free(top_seg,sizeof(OUT_SEG_STRUC));
    if (bot_seg != NULL_SEG)
      mem_free(bot_seg,sizeof(OUT_SEG_STRUC));

    fprintf(stderr," processing near flat triangles
(%u)\n",near_tri_count);
    };
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPTUTL.C                                         */
/*                                                   */
/* PFP Triangle Utilities Module                     */
/*****************************************************/


#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"


extern UINT16 z_control_table [];



/*********************************************************
   exported procedure

   pfp_get_min_z



*********************************************************/


void pfp_get_min_z (tri, min_z)

   INPUT_TRI_STRUC *tri;
   float           *min_z;
```

```
{
   *min_z = (float) (MAX_Z_VAL+1);

   if (tri-> z1 < *min_z)
      *min_z = tri-> z1;

   if (tri-> z2 < *min_z)
      *min_z = tri-> z2;

   if (tri-> z3 < *min_z)
      *min_z = tri-> z3;

} /* pfp_get_min_z */



/*********************************************************
   local procedure

   get_max_z



*********************************************************/

void get_max_z (tri, max_z)

   INPUT_TRI_STRUC *tri;
   float           *max_z;

{
   *max_z = 0.0;
```

```
   if (tri-> z1 > *max_z)
      *max_z = tri-> z1;

   if (tri-> z2 > *max_z)
      *max_z = tri-> z2;

   if (tri-> z3 > *max_z)
      *max_z = tri-> z3;

} /* get_max_z */



/*********************************************************
   local procedure

   order_points

*********************************************************/

void order_points (input_tri)

   INPUT_TRI_STRUC *input_tri;

{
   INPUT_TRI_STRUC *temp_tri;
   float           min_z;
   float           max_z;


   temp_tri = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));
```

```
   pfp_get_min_z (input_tri, &min_z);

   get_max_z (input_tri, &max_z);

   if ((input_tri-> z1 == max_z) && (input_tri-> z2 ==
min_z))
      {
       temp_tri-> x1 = input_tri-> x2;
       temp_tri-> y1 = input_tri-> y2;
       temp_tri-> z1 = input_tri-> z2;

       temp_tri-> x2 = input_tri-> x3;
       temp_tri-> y2 = input_tri-> y3;
       temp_tri-> z2 = input_tri-> z3;

       temp_tri-> x3 = input_tri-> x1;
       temp_tri-> y3 = input_tri-> y1;
       temp_tri-> z3 = input_tri-> z1;
      }

   else if ((input_tri-> z1 == max_z) && (input_tri-> z3
== min_z))
      {
       temp_tri-> x1 = input_tri-> x3;
       temp_tri-> y1 = input_tri-> y3;
       temp_tri-> z1 = input_tri-> z3;

       temp_tri-> x2 = input_tri-> x2;
       t mp_tri-> y2 = input_tri-> y2;
       t mp_tri-> z2 = input_tri-> z2;

       temp_tri-> x3 = input_tri-> x1;
       temp_tri-> y3 = input_tri-> y1;
       temp_tri-> z3 = input_tri-> z1;
      }
```

```
   else if ((input_tri-> z2 == max_z) && (input_tri-> z3
== min_z))
      {
       temp_tri-> x1 = input_tri-> x3;
       temp_tri-> y1 = input_tri-> y3;
       temp_tri-> z1 = input_tri-> z3;

       temp_tri-> x2 = input_tri-> x1;
       temp_tri-> y2 = input_tri-> y1;
       temp_tri-> z2 = input_tri-> z1;

       temp_tri-> x3 = input_tri-> x2;
       temp_tri-> y3 = input_tri-> y2;
       temp_tri-> z3 = input_tri-> z2;
      }

   else if ((input_tri-> z2 == max_z) && (input_tri-> z1
== min_z))
      {
       temp_tri-> x1 = input_tri-> x1;
       temp_tri-> y1 = input_tri-> y1;
       temp_tri-> z1 = input_tri-> z1;

       temp_tri-> x2 = input_tri-> x3;
       temp_tri-> y2 = input_tri-> y3;
       temp_tri-> z2 = input_tri-> z3;

       temp_tri-> x3 = input_tri-> x2;
       temp_tri-> y3 = input_tri-> y2;
       temp_tri-> z3 = input_tri-> z2;
      }

   else if ((input_tri-> z3 == max_z) && (input_tri-> z1
== min_z))
      {
       temp_tri-> x1 = input_tri-> x1;
       temp_tri-> y1 = input_tri-> y1;
```

```
            temp_tri-> z1 = input_tri-> z1;

            temp_tri-> x2 = input_tri-> x2;
            temp_tri-> y2 = input_tri-> y2;
            temp_tri-> z2 = input_tri-> z2;

            temp_tri-> x3 = input_tri-> x3;
            temp_tri-> y3 = input_tri-> y3;
            temp_tri-> z3 = input_tri-> z3;
           }

      else
         {
            temp_tri-> x1 = input_tri-> x2;
            temp_tri-> y1 = input_tri-> y2;
            temp_tri-> z1 = input_tri-> z2;

            temp_tri-> x2 = input_tri-> x1;
            temp_tri-> y2 = input_tri-> y1;
            temp_tri-> z2 = input_tri-> z1;

            temp_tri-> x3 = input_tri-> x3;
            temp_tri-> y3 = input_tri-> y3;
            temp_tri-> z3 = input_tri-> z3;
           }

      temp_tri-> normal_info = input_tri-> normal_info;
      temp_tri-> next_input_tri = input_tri->
   next_input_tri;
      copy(temp_tri, input_tri, sizeof (INPUT_TRI_STRUC));
      mem_free(temp_tri,sizeof(INPUT_TRI_STRUC));

   } /* order_points */
```

```
/*********************************************************
   local procedure

   get_vector



*********************************************************/

void get_vector (vector_ptr, start_x, start_y, start_z,
                        end_x, end_y, end_z)

   struct vector *vector_ptr;
   float start_x;
   float start_y;
   float start_z;
   float end_x;
   float end_y;
   float end_z;

{
   double delta_x, delta_y, delta_z;


   vector_ptr-> x = start_x;
   vector_ptr-> y = start_y;

   delta_z = end_z - start_z;

   if (delta_z > 0.0)
      {
       delta_x = end_x - start_x;
       delta_y = end_y - start_y;
       vector_ptr-> x_incr = delta_x / delta_z;
       vector_ptr-> y_incr = delta_y / delta_z;
```

```
        }
      else
        {
         vector_ptr-> x_incr = 0;
         vector_ptr-> y_incr = 0;
        }

} /* get_vector */



/*************************************************************
   exported procedure

   pfp_get_intersection_segment



*************************************************************/
void pfp_get_intersection_segment (vect_ptr, z_value,
seg_ptr)

   VECT_TRI_STRUC *vect_ptr;
   UINT16         z_value;
   OUT_SEG_STRUC  **seg_ptr;

{
   OUT_SEG_STRUC  *segment;
   double x1,y1,x2,y2;

   if (*seg_ptr == NULL_SEG)
     *seg_ptr = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
   segment = *seg_ptr;
```

```
   segment-> normal_info = vect_ptr-> normal_info;
   x1 = vect_ptr-> vector_1 .x
        + vect_ptr-> vector_1.x_incr * ((double)
(z_value) - vect_ptr->vert1_z);
   y1 = vect_ptr-> vector_1 .y
        + vect_ptr-> vector_1.y_incr * ((double)
(z_value) - vect_ptr->vert1_z);

   if (vect_ptr-> vert2_z > z_value)
      {
       x2 = vect_ptr-> vector_2 .x
            + vect_ptr-> vector_2.x_incr * ((double)
(z_value) - vect_ptr->vert1_z);
       y2 = vect_ptr-> vector_2 .y
            + vect_ptr-> vector_2.y_incr * ((double)
(z_value) - vect_ptr->vert1_z);
      }
   else
      {
       x2 = vect_ptr-> vector_3 .x
                  + vect_ptr-> vector_3.x_incr *
((double) (z_value) - vect_ptr->vert2_z);
       y2 = vect_ptr-> vector_3 .y
                  + vect_ptr-> vector_3.y_incr *
((double) (z_value) - vect_ptr->vert2_z);
      }
   segment->x_1 = round_dbl(x1);
   segment->y_1 = round_dbl(y1);
   segment->x_2 = round_dbl(x2);
   segment->y_2 = round_dbl(y2);

} /* pfp_get_intersection_segment */


/*******************************************************
   exported procedure
```

```
*****************************************************/

void pfp_convert_tri_to_vect(input_tri, vect_tri)

   INPUT_TRI_STRUC *input_tri;
   VECT_TRI_STRUC  *vect_tri;

{
   struct vector   *vector_ptr;


   order_points (input_tri);

   vector_ptr = (struct vector *) vect_tri;

   get_vector (vector_ptr++,
               input_tri-> x1, input_tri-> y1,
input_tri-> z1,
               input_tri-> x3, input_tri-> y3,
input_tri-> z3);

   get_vector (vector_ptr++,
               input_tri-> x1, input_tri-> y1,
input_tri-> z1,
               input_tri-> x2, input_tri-> y2,
input_tri-> z2);

   get_vector (vector_ptr,
               input_tri-> x2, input_tri-> y2,
input_tri-> z2,
               input_tri-> x3, input_tri-> y3,
input_tri-> z3);

   vect_tri-> vert1_z  = input_tri->z1;
   vect_tri-> vert2_z  = input_tri->z2;
```

```
   vect_tri-> vert3_z  = input_tri->z3;
   vect_tri-> normal_info = input_tri-> normal_info;
   vect_tri-> next_tri = NULL_VECT_TRI;

} /* pfp_convert_tri_to_vect */

/*********************************************************
   procedure


*********************************************************/


void pfp_convert_tri_to_segs (tri_ptr, seg_list)

   INPUT_TRI_STRUC *tri_ptr;
   OUT_SEG_STRUC   **seg_list;

{
   OUT_SEG_STRUC   *seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
   seg_ptr-> x_1 = round_flt(tri_ptr-> x1);
   seg_ptr-> y_1 = round_flt(tri_ptr-> y1);
   seg_ptr-> x_2 = round_flt(tri_ptr-> x2);
   seg_ptr-> y_2 = round_flt(tri_ptr-> y2);
   seg_ptr->normal_info = tri_ptr->normal_info;
   pfp_get_xy_normal_info (tri_ptr-> x1, tri_ptr-> y1,
                           tri_ptr-> x2, tri_ptr-> y2,
                           tri_ptr-> x3, tri_ptr->
y3,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
```

```
   seg_ptr-> x_1 = round_flt(tri_ptr-> x2);
   seg_ptr-> y_1 = round_flt(tri_ptr-> y2);
   seg_ptr-> x_2 = round_flt(tri_ptr-> x3);
   seg_ptr-> y_2 = round_flt(tri_ptr-> y3);
   seg_ptr->normal_info = tri_ptr->normal_info;
   pfp_get_xy_normal_info (tri_ptr-> x2, tri_ptr-> y2,
                           tri_ptr-> x3, tri_ptr-> y3,
                           tri_ptr-> x1, tri_ptr->
y1,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

   seg_ptr = (OUT_SEG_STRUC *) mem_alloc (sizeof
(OUT_SEG_STRUC));
   seg_ptr-> x_1 = round_flt(tri_ptr-> x3);
   seg_ptr-> y_1 = round_flt(tri_ptr-> y3);
   seg_ptr-> x_2 = round_flt(tri_ptr-> x1);
   seg_ptr-> y_2 = round_flt(tri_ptr-> y1);
   seg_ptr->normal_info = tri_ptr->normal_info;
   pfp_get_xy_normal_info (tri_ptr-> x3, tri_ptr-> y3,
                           tri_ptr-> x1, tri_ptr-> y1,
                           tri_ptr-> x2, tri_ptr->
y2,&seg_ptr->normal_info);
   seg_ptr-> next_seg = *seg_list;
   *seg_list = seg_ptr;

} /* pfp_convert_tri_to_segs */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPUTL.C                                          */
/*                                                   */
/* PFP Utilities module                              */
/*****************************************************/

#include <stdio.h>
#include <time.h>
#include <malloc.h>
#include <math.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
/*#include "pfputl.h"*/

static long   last_time;

extern long max_mem_alloc;
extern long total_mem_alloc;
extern long curr_mem_alloc;

float read_float(infile,items)
  FILE *infile;
  INT16 *items;
  {
  float tempfloat;
  char  *floatptr,tempchar;

  if (fread(&tempfloat,4,1,infile)==1) (*items)++;

  /* if IRIS sytem, revers floating point numbers */
#ifdef mips
  floatptr = (char *) &tempfloat;
  tempchar = *floatptr;
```

```
  *floatptr = *(floatptr+3);
  *(floatptr+3) = tempchar;
  tempchar = *(floatptr+1);
  *(floatptr+1) = *(floatptr+2);
  *(floatptr+2) = tempchar;
#endif

  return(tempfloat);
  };


double round_dbl(dbl_val)
  double dbl_val;
  {
  if (dbl_val < 0.0) dbl_val = ceil(0.5+dbl_val);
  else dbl_val = floor(0.5+dbl_val);
  return (dbl_val);

} /* round_dbl */

float round_flt(float_val)
  float float_val;
  {
  return ((float) round_dbl((double) float_val));

} /* round_flt */

INT16 sign(d)
  double d;
  {
  if (d < 0.0) return(-1);
  else return(1);
  };


STRING *mem_alloc (size)
```

```
   UINT32 size;

{
   STRING *ptr;

   ptr = (STRING *) malloc (size);

   curr_mem_alloc += size;
   total_mem_alloc += size;

   if (curr_mem_alloc > max_mem_alloc)
      max_mem_alloc = curr_mem_alloc;

   if (ptr == (STRING *) NULL)
      {
       fprintf (stderr, "\n error - memory allocation
failed\n");
       fprintf (stderr, "\n current allocation = %ld\n",
curr_mem_alloc);
       exit (SYS_FAILURE);
      }

   return (ptr);

} /* mem_alloc */



void mem_free (ptr,num_bytes)

   STRING  *ptr;
   UINT32  num_bytes;

{

   if (ptr != (STRING *) NULL)
      {
```

```
        curr_mem_alloc -= num_bytes;
        free (ptr);
       }
   else
      {
       fprintf (stderr, "\n error - null pointer freed
\n");
       exit (SYS_FAILURE);
      }

} /* mem_free */


void get_timestamp (timestamp)

   STRING *timestamp;

{
   long time_buf;

   time (&time_buf);
   strcpy (timestamp, ctime (&time_buf));

} /* get_timestamp */









void openfile (filename, file_ptr, open_mode)

   STRING *filename;
   FILE   **file_ptr;
   STRING *open_mode;

{
```

```
   if (*filename == '\0')
      *file_ptr = (FILE *) NULL;
   else
      {
       *file_ptr = fopen (filename, open_mode);
       if (*file_ptr == (FILE *) NULL)
          {
           fprintf (stderr, "\n error - unable to open
%s\n", filename);
           exit (SYS_FAILURE);
          }
      }

} /* openfile */



void copy (source_ptr, dest_ptr, byte_count)

   STRING *source_ptr;
   STRING *dest_ptr;
   INT16  byte_count;

{
   INT16 i;

   for (i = 0; i < byte_count; i++)
      {
       *dest_ptr = *source_ptr;
       dest_ptr++;
       source_ptr++;
      }

} /* copy */
```

```
void pfp_get_xy_normal_info
(x1,y1,x2,y2,x3,y3,normal_info)
  float x1;
  float y1;
  float x2;
  float y2;
  float x3;
  float y3;
  UINT8 *normal_info;
  {
  double slope, offset;

  *normal_info &=
(255-NORMAL_X_NEGATIVE-NORMAL_X_POSITIVE-

NORMAL_Y_NEGATIVE-NORMAL_Y_POSITIVE);

  if ((x2 - x1) != 0.0) slope = (y2 - y1) / (x2 - x1);
  else slope = 0.0;

  offset = y1 - slope*x1;

  if (slope == 0.0) {
    if (y1 == y2) {
      if (y3 < y2) *normal_info |= NORMAL_Y_POSITIVE;
      else *normal_info |= NORMAL_Y_NEGATIVE;
      };
    if (x1 == x2) {
      if (x3 < x2) *normal_info |= NORMAL_X_POSITIVE;
      else *normal_info |= NORMAL_X_NEGATIVE;
      };
    }
  else {
    if (((y3 - offset)/slope) >= x3) *normal_info |=
NORMAL_X_POSITIVE;
    else *normal_info |= NORMAL_X_NEGATIVE;
```

```
    if ((slope*x3 + offset) >= y3) *normal_info |=
NORMAL_Y_POSITIVE;
    else *normal_info |= NORMAL_Y_NEGATIVE;
    }
  };
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPGRAF.C                                         */
/*                                                   */
/* PFP graphics module                               */
/*****************************************************/

#include "pfp.h"

#define Z_LABEL_Z_POSITION 5500
#define Z_LABEL_X_POSITION MAX_X_VAL + 3000

#define Z_VAL_Z_POSITION 50
#define Z_VAL_X_POSITION MAX_X_VAL + 2000

#define LOOP_VAL_Z_POSITION 50000
#define LOOP_VAL_X_POSITION MAX_X_VAL + 4000

#define HORIZ_MARGIN 17000
#define LEFT_EDGE_COORD  0.0 - HORIZ_MARGIN
#define RIGHT_EDGE_COORD MAX_X_VAL + HORIZ_MARGIN




void init_graphics ()

{
/*
   pBegin ("ibm_cga");
*/
/* ??
   pBegin ("hgc");
```

```
   pSetActivePage (0);
   pSetDisplayPage (0);
   pDefineWindow ((double) LEFT_EDGE_COORD, (double) 0,
                  (double) RIGHT_EDGE_COORD, (double)
MAX_Y_VAL);
   pDefineViewPort ((double) 0, (double) 0, (double)
1.0, (double) 1.0);
   pAllowMappingDistortion (1);
   pEnableWindowClipping (0);
   pSetColor (1);
   pSetCharSize (1, 2);

?? */
} /* init_graphics */



void pfp_display_section_code (section_code)

   UINT8 section_code;

{
   STRING code_string [10];

   /* ??
   sprintf (code_string, "%c", section_code);

   pMove ((double) LOOP_VAL_X_POSITION, (double)
LOOP_VAL_Z_POSITION);
   pString (code_string);
   ?? */
} /* pfp_display_section_code */



void setup_screen (z_val, loop_count)
```

```
  INT16 z_val;
  INT16 loop_count;

{
   STRING string_buf [100];

/*
   if (debug_mode)
      pPause (1);
*/
   /* ??
   pSetColor (1);
   pDrawRectangle ((double) 0.0, (double) 0.0, (double)
MAX_X_VAL,
                   (double) MAX_Y_VAL);
   pMove ((double) Z_LABEL_X_POSITION, (double)
Z_LABEL_Z_POSITION);
   pString ("Z");
   pMove ((double) Z_VAL_X_POSITION, (double)
Z_VAL_Z_POSITION);
   sprintf (string_buf, "%d", z_val);
   pString (string_buf);

   pMove ((double) LOOP_VAL_X_POSITION, (double)
LOOP_VAL_Z_POSITION);
   sprintf (string_buf, "%d", loop_count);
   pString (string_buf);
   ?? */
} /* setup_screen */


void draw_line (x1, z1, x2, z2)

   UINT16       x1, z1, x2, z2;
```

```
{
   /* ??
   pMove ((double) (UINT16) x1, (double) (UINT16) z1);
   pDrawLine ((double) (UINT16) x2, (double) (UINT16)
z2);
   ??? */
} /* draw_line */



void set_display_mode (display_mode)

  INT16 display_mode;

{
   /* ?? pSetColor (display_mode);  ?? */

} /* set_display_mode */



void end_graphics ()

{
   /* ?? pEnd ();  ?? */

} /* end_graphics */
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PFPHI.C                                             */
/*                                                     */
/* PFP human interface module                          */
/*******************************************************/


/*******************************************************

   PFPHI.C contains the following routines:

   get_arg_code..... converts command line arg
string to code
   not_a_number..... checks a string for a legal
numeric value
   get_float_arg..... processes a command line arg
float parameter
   get_int_arg..... processes a command line arg
integer parameter
   get_filename_arg..... processes an arg that has a
filename parameter
   get_user_info_cmd_line.. gets all user info from the
command line
   init_user_info..... initializes user_info
structure to standard defaults
   print_cmd_line_help..... prints command line help
text to the screen
   check_user_info..... retrieves and processes user
option input
   get_user_info..... retrieves and processes user
option input

*******************************************************/
```

```
#include <stdio.h>

#include "pfp.h"
#include "pfphi.h"



/*********************************************************
   local procedure

   get_arg_code

   Takes a command line arg and returns the code value
of the arg or an unknown
   arg code value.

*********************************************************/


static void get_arg_code (input_arg, arg_code)

   STRING *input_arg;
   INT16  *arg_code;

{
   STRING arg[100];
   INT16  i;


   strcpy (arg, input_arg);

   for (i = 0; i < strlen (arg); i++)
      arg [i] = toupper (arg [i]);
```

```
if (!strcmp (arg, RES_OPTION))
   *arg_code = RES_OPTION_CODE;

else if (!strcmp (arg, HELP_OPTION))
   *arg_code = HELP_OPTION_CODE;

else if (!strcmp (arg, SEGFILE_OPTION))
   *arg_code = SEGFILE_OPTION_CODE;

else if (!strcmp (arg, DISPLAY_OPTION))
   *arg_code = DISPLAY_OPTION_CODE;

else if (!strcmp (arg, Z_SPACE_OPTION))
   *arg_code = Z_SPACE_OPTION_CODE;

else if (!strcmp (arg, Z_SPACE_FILE_OPTION))
   *arg_code = Z_SPACE_FILE_OPTION_CODE;

else if (!strcmp (arg, XF_HATCH_SPACE_OPTION))
   *arg_code = XF_HATCH_SPACE_OPTION_CODE;


else if (!strcmp (arg, YF_HATCH_SPACE_OPTION))
   *arg_code = YF_HATCH_SPACE_OPTION_CODE;


else if (!strcmp (arg, Y_HATCH_SPACE_OPTION))
   *arg_code = Y_HATCH_SPACE_OPTION_CODE;


else if (!strcmp (arg, X_HATCH_SPACE_OPTION))
   *arg_code = X_HATCH_SPACE_OPTION_CODE;


else if (!strcmp (arg, ANGLE_HATCH_SPACE_OPTION))
   *arg_code = ANGLE_HATCH_SPACE_OPTION_CODE;
```

```
else if (!strcmp (arg, CENTER_OPTION))
   *arg_code = CENTER_OPTION_CODE;

else if (!strcmp (arg, SECTION_CODES_OFF_OPTION))
   *arg_code = SECTION_CODES_OFF_OPTION_CODE;


else if (!strcmp (arg, MIN_SURFACE_ANGLE_OPTION))
   *arg_code = MIN_SURFACE_ANGLE_OPTION_CODE;


else if (!strcmp (arg, MIN_INTERSECT_ANGLE_OPTION))
   *arg_code = MIN_INTERSECT_ANGLE_OPTION_CODE;


else if (!strcmp (arg, X_AXIS_OPTION))
   *arg_code = X_AXIS_OPTION_CODE;

else if (!strcmp (arg, Y_AXIS_OPTION))
   *arg_code = Y_AXIS_OPTION_CODE;

else if (!strcmp (arg, BINARY_INPUT_OPTION))
   *arg_code = BINARY_INPUT_OPTION_CODE;

else if (!strcmp (arg, DEBUG_OPTION))
   *arg_code = DEBUG_OPTION_CODE;

else if (!strcmp (arg, DISK_OPTION))
   *arg_code = DISK_OPTION_CODE;

else if (!strcmp (arg, ARIES_OPTION))
   *arg_code = ARIES_OPTION_CODE;

else if (!strcmp (arg, CURE_OPTION))
   *arg_code = CURE_OPTION_CODE;
```

```
   else if (!strcmp (arg, INTERNAL_OPTION))
      *arg_code = INTERNAL_OPTION_CODE;

   else
      *arg_code = UNKNOWN_OPTION_CODE;

} /* get_arg_code */



/*******************************************************
   local function

   not_a_number

   Takes a character string and determines if it
represents a legal integer or
   real value.  A legal number contains only the chars
'0' through '9' with at
   most one decimal point and possibly a '-' sign at the
beginning of the
   string.

*******************************************************/


static INT16 not_a_number (number_string)

   STRING  *number_string;

{
   BOOLEAN decimal_point_found = FALSE;
   BOOLEAN not_a_num = FALSE;

   short number_length;
```

```
   short i = 0;


   number_length = strlen (number_string);

   if (number_length == 0)
      not_a_num = TRUE;

   else if ((number_string[i] == '-') ||
(number_string[i] == '.'))
      {
       i++;
       not_a_num = (i == number_length);
      }

   while ((i < number_length) && !not_a_num)
      {
       if (number_string[i] == '.')
          {
           not_a_num = decimal_point_found;
           decimal_point_found = TRUE;
          }
       else
           not_a_num = ((number_string[i] < '0') ||
(number_string[i] > '9'));
       i++;
      }

   return ((short) not_a_num);

} /* not_a_number */
```

```
/*********************************************************
   local procedure

   get_float_arg

   Gets the float parameter of the command line option
identified by
   option_name and arg_index.
   Checks for error. Increments arg_index if
appropriate.

*********************************************************/


static void get_float_arg (float_val, argc, argv,
arg_index, option_name,
                           error)

   float   *float_val; /* must be float for scanf */
   INT16   argc;
   STRING  *argv[];
   INT16   *arg_index;
   STRING  *option_name;
   BOOLEAN *error;

{
   short arg_code;


   if ((*arg_index + 1) == argc)
      {
       *error = TRUE;
       fprintf (stderr, "\n error - missing parameter
for %s option \n",
                option_name);
      }
   else
```

```
      {
       if (not_a_number (argv [*arg_index + 1]))
          {
           *error = TRUE;
           get_arg_code (argv [*arg_index + 1],
&arg_code);
           if (arg_code == UNKNOWN_OPTION_CODE)
              {
               fprintf (stderr, "\n error - invalid
parameter for %s option \n",
                           option_name);
               (*arg_index)++ ;
              }
           else
              fprintf (stderr, "\n error - missing
parameter for %s option \n",
                           option_name);
          }
       else
          {
           if (strlen (argv [*arg_index + 1]) >
MAX_FLOAT_LENGTH)
              {
               *error = TRUE;
               fprintf (stderr, " \n error - real number
length exceeds limit");
               fprintf (stderr, " for %s option \n",
option_name);
              }
           else
              sscanf (argv [*arg_index + 1], "%f",
float_val);

           (*arg_index)++ ;
          }
      }
```

```
} /* get_float_arg */


/*********************************************************
   local procedure

   get_int_arg

   Gets the int parameter of the command line option
identified by
   option_name and arg_index.
   Checks for error. Increments arg_index if
appropriate.

*********************************************************/


static void get_int_arg (int_val, argc, argv, arg_index,
option_name, error)

   INT16   *int_val;
   INT16   argc;
   STRING  *argv[];
   INT16   *arg_index;
   STRING  *option_name;
   BOOLEAN *error;

{
   short arg_code;
   int   temp_int; /* must be int for scanf */


   if ((*arg_index + 1) == argc)
```

```
        {
         *error = TRUE;
         fprintf (stderr, "\n error - missing parameter
for %s option \n",
                  option_name);
        }
     else
        {
         if (not_a_number (argv [*arg_index + 1]))
            {
             *error = TRUE;
             get_arg_code (argv [*arg_index + 1],
&arg_code);
             if (arg_code == UNKNOWN_OPTION_CODE)
                {
                 fprintf (stderr, "\n error - invalid
parameter for %s option \n",
                                option_name);
                 (*arg_index)++ ;
                }
             else
                fprintf (stderr, "\n error - missing
parameter for %s option \n",
                                option_name);
            }
         else
            {
             if (strlen (argv [*arg_index + 1]) >
MAX_INT_LENGTH)
                {
                 *error = TRUE;
                 fprintf (stderr, " \n error - number
length exceeds limit");
                 fprintf (stderr, " for %s option \n",
option_name);
                }
             else {
```

```
                sscanf (argv [*arg_index + 1], "%d",
&temp_int);
                *int_val = temp_int;
                };
             (*arg_index)++ ;
            }
        }

} /* get_int_arg */



/*******************************************************
   local procedure

   get_filename_arg

   Gets the filename parameter of the command line
option identified by
   option_name and arg_index and copies it to the
filename parameter.
   Checks for error.
   Increments arg_index if appropriate.

*******************************************************/


static void get_filename_arg (filename, argc, argv,
arg_index  option_name,
                              error)

   STRING  *filename;
   INT16   argc;
   STRING  *argv[];
   short   *arg_index;
   STRING  *option_name;
```

```
   BOOLEAN *error;

{
   short arg_code;

   if ((*arg_index + 1) == argc)
      {
       *error = TRUE;
       printf ("\n error - missing filename parameter
for %s option \n",
                option_name);
      }
   else
      {
       get_arg_code (argv [*arg_index + 1], &arg_code);

       if (arg_code != UNKNOWN_OPTION_CODE)
          {
           *error = TRUE;
           printf ("\n error - missing filename
parameter for %s option \n",
                    option_name);
          }

       else
          {
           strcpy (filename, argv [*arg_index + 1]);
           (*arg_index)++ ;
          }
      }

} /* get_filename_arg */
```

```
/*********************************************************
   local procedure

   get_user_info_cmd_line

   Reads the command line information and stores the
data into the structure
   user_info.
   Checks for error.

*********************************************************/


static void get_user_info_cmd_line (user_info, argc,
argv, cmd_line_help_needed,

cmd_line_help_requested)

   USER_INFO_STRUC *user_info;
   INT16           argc;
   STRING          *argv[];
   BOOLEAN         *cmd_line_help_needed;
   BOOLEAN         *cmd_line_help_requested;

{
   float  float_arg;
   INT16  int_arg;
   STRING filename_arg [100];
   INT16  arg_index,arg_code;


   *cmd_line_help_needed = FALSE;
   *cmd_line_help_requested = FALSE;

   arg_index = 1;
```

```
   while (arg_index < argc)
      {
       get_arg_code (argv[arg_index], &arg_code);

       switch (arg_code)
          {
           case RES_OPTION_CODE :
              get_float_arg (&float_arg, argc, argv,
&arg_index,
                             RES_OPTION,
cmd_line_help_needed);
              user_info-> res_value = float_arg;
              if (user_info->res_value <= 0.0) {
                fprintf (stderr,"\n error - res value
out of range - %-1.3f \n", user_info->res_value);
                exit(SYS_FAILURE);
                }
              break;

           case Z_SPACE_OPTION_CODE :
              get_int_arg (&int_arg, argc, argv,
&arg_index,
                           Z_SPACE_OPTION,
cmd_line_help_needed);
              user_info->z_spacing = int_arg;
              if (user_info->z_spacing <= 0) {
                fprintf (stderr,"\n error - z spacing
value out of range - %u \n", user_info->z_spacing);
                exit(SYS_FAILURE);
                }
              break;

           case YF_HATCH_SPACE_OPTION_CODE :
              get_int_arg (&int_arg, argc, argv,
&arg_index,
                           YF_HATCH_SPACE_OPTION,
cmd_line_help_needed);
```

```
                user_info-> y_flat_fill_spacing = int_arg;

                break;

            case XF_HATCH_SPACE_OPTION_CODE :
                get_int_arg (&int_arg, argc, argv,
&arg_index,
                             XF_HATCH_SPACE_OPTION,
cmd_line_help_needed);
                user_info-> x_flat_fill_spacing = int_arg;

                break;

            case Y_HATCH_SPACE_OPTION_CODE :
                get_int_arg (&int_arg, argc, argv,
&arg_index,
                             Y_HATCH_SPACE_OPTION,
cmd_line_help_needed);
                user_info-> y_hatch_spacing = int_arg;
                break;

            case X_HATCH_SPACE_OPTION_CODE :
                get_int_arg (&int_arg, argc, argv,
&arg_index,
                             X_HATCH_SPACE_OPTION,
cmd_line_help_needed);
                user_info-> x_hatch_spacing = int_arg;
                break;

            case ANGLE_HATCH_SPACE_OPTION_CODE :
                get_int_arg (&int_arg, argc, argv,
&arg_index,
                             ANGLE_HATCH_SPACE_OPTION,
cmd_line_help_needed);
                user_info-> angle_hatch_spacing = int_arg;

                break;
```

```
            case MIN_SURFACE_ANGLE_OPTION_CODE :
               get_int_arg (&int_arg, argc, argv,
&arg_index,
                                 MIN_SURFACE_ANGLE_OPTION,
cmd_line_help_needed);
               user_info-> min_surface_angle = int_arg;
               break;

            case MIN_INTERSECT_ANGLE_OPTION_CODE :
               get_int_arg (&int_arg, argc, argv,
&arg_index,
                                 MIN_INTERSECT_ANGLE_OPTION,
cmd_line_help_needed);
               user_info-> min_intersect_angle = int_arg;

               break;

            case X_AXIS_OPTION_CODE :
               user_info-> axis = 1;
               break;

            case Y_AXIS_OPTION_CODE :
               user_info-> axis = 2;
               break;

            case BINARY_INPUT_OPTION_CODE :
               user_info-> binary_input = TRUE;
               break;

            case SEGFILE_OPTION_CODE :
               get_filename_arg (filename_arg, argc,
argv, &arg_index,
                                    SEGFILE_OPTION,
cmd_line_help_needed);
               strcpy (user_info-> segfile_name,
filename_arg);
```

```
                break;

            case Z_SPACE_FILE_OPTION_CODE :
                get_filename_arg (filename_arg, argc,
argv, &arg_index,
                                  Z_SPACE_OPTION,
cmd_line_help_needed);
                strcpy (user_info-> z_space_filename,
filename_arg);
                break;

            case HELP_OPTION_CODE :
                *cmd_line_help_requested = TRUE;
                break;

            case DEBUG_OPTION_CODE :
                user_info-> debug_mode = TRUE;
                break;

            case CENTER_OPTION_CODE :
                user_info-> centering_off = FALSE;
                break;

            case SECTION_CODES_OFF_OPTION_CODE :
                user_info-> section_codes_off = TRUE;
                break;

            case DISPLAY_OPTION_CODE :
                user_info-> display_on = TRUE;
                break;

            case DISK_OPTION_CODE :
                user_info-> in_mem = FALSE;
                break;

            case ARIES_OPTION_CODE :
                user_info-> binary_input = TRUE;
```

```
                user_info-> attrib = FALSE;
                break;

            case INTERNAL_OPTION_CODE :
                get_filename_arg (filename_arg, argc,
argv, &arg_index,
                                    INTERNAL_OPTION,
cmd_line_help_needed);
                strcpy (user_info-> internal_filename,
filename_arg);
                break;

            case CURE_OPTION_CODE :
                get_int_arg (&int_arg, argc, argv,
&arg_index,
                                CURE_OPTION,
cmd_line_help_needed);
                user_info-> cure = int_arg;
                if (user_info->cure <= 0) {
                  fprintf (stderr,"\n error - cure value
out of range - %d \n", user_info->cure);
                  exit(SYS_FAILURE);
                  }
                break;

            case UNKNOWN_OPTION_CODE :
                {
                 *cmd_line_help_needed = TRUE;
                 fprintf (stderr,
                      "\n error - unknown option - %s \n",
argv [arg_index]);
                }
            }
         arg_index++ ;
        }
```

```
} /* get_user_info_cmd_line */


/*********************************************************
   local procedure

   init_user_info

   Initializes user_info structure to default values.

*********************************************************/

static void init_user_info (user_info)

   USER_INFO_STRUC *user_info;

{
   user_info-> res_value = DEFAULT_RES_VALUE;
   strcpy (user_info-> segfile_name,
DEFAULT_SEGFILE_NAME);
   user_info-> debug_mode = FALSE;
   user_info-> display_on = FALSE;
   user_info-> centering_off = TRUE;
   user_info-> section_codes_off = FALSE;
   user_info-> axis = 0;
   user_info-> binary_input = FALSE;
   user_info-> attrib = TRUE;
   user_info-> in_mem = TRUE;
   user_info-> cure = 0;
   user_info-> z_spacing = 1;
   user_info-> y_flat_fill_spacing = 0;
   user_info-> x_flat_fill_spacing = 0;
```

```
   user_info-> y_hatch_spacing = 0;
   user_info-> x_hatch_spacing = 0;
   user_info-> angle_hatch_spacing = 0;
   user_info-> min_surface_angle = 0;
   user_info-> min_intersect_angle = 0;
   *(user_info-> z_space_filename) = '\0';
   *(user_info-> internal_filename) ='\0';

} /* init_user_info */




/*********************************************************
   local procedure

   print_cmd_line_help

   Prints command line help message explaining command
line syntax.

*********************************************************/


static void print_cmd_line_help ()

{
fprintf (stderr, "\n PFP Command Line Syntax: \n");
fprintf (stderr, "\n PFP < input_file     <options>
\n");
fprintf (stderr, "\n   where <options> indicates any of
the following:\n");
fprintf (stderr, "\n     -res <res_value>");
fprintf (stderr, "\n        Used to specify a real
number resolution value that");
```

```
fprintf (stderr, "\n        will be used to scale the
input file coordinates.");
fprintf (stderr, "\n        The default resolution value
is %f\n",

DEFAULT_RES_VALUE);
fprintf (stderr, "\n     -sf <segment output file>");
fprintf (stderr, "\n        Used to specify an output
file for");
fprintf (stderr, " z plane intersection segments.");
fprintf (stderr, "\n        The default is to not
generate a segment");
fprintf (stderr, " output file\n");
fprintf (stderr, "\n     -zs <z spacing value>");
fprintf (stderr, "\n        Used to indicate the
constant z spacing of the");
fprintf (stderr, " output cross-sections.\n");
fprintf (stderr, "\n     -hy <hatch spacing value>");
fprintf (stderr, "\n        Used to set spacing of hatch
lines parallel");
fprintf (stderr, " to the y axis.\n");
fprintf (stderr, "\n     -hx <hatch spacing value>");
fprintf (stderr, "\n        Used to set spacing of hatch
lines parallel");
fprintf (stderr, " to the x axis.\n");
fprintf (stderr, "\n     -ha <hatch spacing value>");
fprintf (stderr, "\n        Used to set spacing of 60
degree angle");
fprintf (stderr, " hatch lines.\n");
fprintf (stderr, "\n     -hfy <hatch spacing value>");
fprintf (stderr, "\n        Sets spacing of hatch lines
parallel");
fprintf (stderr, " to the y axis on flat surfaces.\n");
fprintf (stderr, "\n     -hfx <hatch spacing value>");
fprintf (stderr, "\n        Sets spacing of hatch lines
parallel");
fprintf (stderr, " to the x axis on flat surfaces.\n");
```

```
fprintf (stderr, "\n     -zsf <z spacing control
file>");
fprintf (stderr, "\n        Used to specify the file
containing variable");
fprintf (stderr, " z spacing information.\n");
fprintf (stderr, "\n     -c");
fprintf (stderr, "\n        Centering option.");
fprintf (stderr, "\n        Center object
coordinates\n");
fprintf (stderr, "\n     -nsc");
fprintf (stderr, "\n        No section codes option.");
fprintf (stderr, "\n        Suppresses section codes in
the output file\n");
fprintf (stderr, "\n     -msa");
fprintf (stderr, "\n        Minimum surface angle
option.");
fprintf (stderr, "\n        Indicates minimum surface
angle for scanned");
fprintf (stderr, "\n        facets.  Legal values are
between 0 and 90"); fprintf (stderr, " degrees.\n");
fprintf (stderr, "\n     -mia");
fprintf (stderr, "\n        Minimum intersect angle
option.");
fprintf (stderr, "\n        Indicates minimum intersect
angle for hatch");
fprintf (stderr, "\n        lines.  Legal values are
between 0 and 90");
fprintf (stderr, " degrees.\n");
fprintf (stderr, "\n     -b");
fprintf (stderr, "\n        Binary input option.");
fprintf (stderr, "\n        Indicates that the input
file is in binary");
fprintf (stderr, " attributed format.");
fprintf (stderr, "\n     -d");
fprintf (stderr, "\n        Display option.");
fprintf (stderr, "\n        Causes the program to
display cross-sections");
```

```
fprintf (stderr, "\n        on the screen as they are
generated.\n");
fprintf (stderr, "\n     -h");
fprintf (stderr, "\n        Help option.");
fprintf (stderr, "\n        Causes the program to print
this help");
fprintf (stderr, " message and then stop.\n");
fprintf (stderr, "\n     -disk");
fprintf (stderr, "\n        Disk option.");
fprintf (stderr, "\n        Causes the program save
layers on disk rather");
fprintf (stderr, "\n        than in memory.  Use for
large files.\n");
fprintf (stderr, "\n     -x or -y");
fprintf (stderr, "\n        Slice Axis option.");
fprintf (stderr, "\n        Changes the Slice axis from
the Z to X or Y\n");
fprintf (stderr, "\n     -internal <triangle analysis
file>");
fprintf (stderr, "\n        Generates a special binary
file consisting of");
fprintf (stderr, "\n        Slice's own internal
representation of triangles,");
fprintf (stderr, "\n        useful for analysis
purposes\n");
fprintf (stderr, "\n     -cure <cure value>");
fprintf (stderr, "\n        Select Cure Compensation.
Cure value is amount");
fprintf (stderr, "\n        to compensate\n");
fprintf (stderr, "\n");

} /* print_cmd_line_help */
```

```
/*******************************************************
   local procedure

   check_user_info

   Checks user_info structure for legal values.
   Sets display_on option if not set and no segment file
was specified.

*******************************************************/


static void check_user_info (user_info,
cmd_line_help_needed)

   USER_INFO_STRUC *user_info;
   BOOLEAN         *cmd_line_help_needed;

{
   if (!user_info-> display_on && (*(user_info->
segfile_name) == '\0'))

       user_info-> display_on = TRUE;

   if ((user_info-> min_surface_angle < 0) ||
       (user_info-> min_surface_angle > MAX_ANGLE))
      {
       fprintf (stderr,
       "\n error - minimum surface angle (%d) must be
between 0 and %d \n",
                user_info-> min_surface_angle,
MAX_ANGLE);
       *cmd_line_help_needed = TRUE;
      }
   if ((user_info-> min_intersect_angle < 0) ||
       (user_info-> min_intersect_angle > MAX_ANGLE))
```

```
        {
         fprintf (stderr,
         "\n error - minimum intersect angle (%d) must be
between 0 and %d \n",
                  user_info-> min_intersect_angle,
MAX_ANGLE);
         *cmd_line_help_needed = TRUE;
        }

} /* check_user_info */




/*********************************************************
   exported procedure

   pfp_get_user_info

   Gets user input from the command line.
   Returns user input in the structure user_info.

*********************************************************/


void pfp_get_user_info (argc, argv, user_info)

   INT16           argc;
   STRING          *argv[];
   USER_INFO_STRUC *user_info;

{
   BOOLEAN cmd_line_help_needed,
cmd_line_help_requested;
```

```
   init_user_info (user_info);

   get_user_info_cmd_line (user_info, argc, argv,
&cmd_line_help_needed,
                           &cmd_line_help_requested);

   check_user_info (user_info, &cmd_line_help_needed);

   if (cmd_line_help_requested)
      {
       print_cmd_line_help ();
       exit (SYS_FAILURE);
      }
   else if (cmd_line_help_needed)
      {
       fprintf (stderr, "\n -- for command line help try
the %s option \n",
                           HELP_OPTION);
       exit (SYS_FAILURE);
      }

} /* pfp_get_user_info */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPIDB.C                                          */
/*                                                   */
/* PFP Input Database Module                         */
/*****************************************************/


#include <stdio.h>
#include <math.h>
#include <malloc.h>

#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"


extern INPUT_TRI_STRUC *tri_bucket_array[];
extern OUT_SEG_STRUC *FD_seg_bucket_array[];
extern OUT_SEG_STRUC *FU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFU_seg_bucket_array[];
extern OUT_SEG_STRUC *NFD_seg_bucket_array[];
extern UINT16        z_control_table[];
extern UINT16        all_min_z, all_max_z,
all_min_z_index, all_max_z_index;
extern FILE          *message_file;

/*******************************************************
   general procedure

   find_z_binarily - binary search of z_control_table
for layer z
```

```
**********************************************************/

UINT16 get_z_binarily(z)

   UINT16 z;

{
   UINT16 low = 0, mid, high = MAX_Z_INDEX-1;

   while (low <= high) {
      mid = (low + high) / 2;
      if (z < z_control_table [mid])
         high = mid - 1;
      else if (z > z_control_table [mid])
         low = mid + 1;
      else
         return(mid);
    }
    if (z_control_table [low] > z)
       return(low);
    else return(high);

}  /* get_z_binarily */


/*********************************************************
   local procedure

   get_offsets


**********************************************************/
```

```
static void get_offsets (min_x, min_y, min_z, max_x,
max_y, max_z,
                         res_value, x_offset, y_offset,
z_offset)

   float min_x, min_y, min_z, max_x, max_y, max_z,
res_value;
   float *x_offset, *y_offset, *z_offset;

{
   BOOLEAN res_error1,res_error2;
   float   x_range, y_range, z_range;
   float   round_flt();


   x_range = (max_x - min_x) * res_value;
   y_range = (max_y - min_y) * res_value;
   z_range = (max_z - min_z) * res_value;

   res_error1 = ((x_range > MAX_X_VAL) || (y_range >
MAX_Y_VAL) ||
                 (z_range > MAX_Z_VAL));

   res_error2 = (z_range > z_control_table
[MAX_Z_INDEX-1]);

   if (res_error1 || res_error2) {
      if (res_error1) {
         fprintf (stderr, "\n error - resolution value
too large for this");
         fprintf (stderr, " part\n");
      }
      if (res_error2) {
         fprintf (stderr, "\n error - z-value would
require more than");
         fprintf (stderr, " %u layers.\n",MAX_Z_INDEX);
      }
```

```
      fprintf (stderr,
      " min x = %f max x = %f min y = %f max y = %f min
z = %f max z = %f\n",
        min_x, max_x, min_y, max_y, min_z, max_z);
      exit (SYS_FAILURE);
   }

   *x_offset = round_flt(((MAX_X_VAL - x_range) / 2.0) -
(min_x * res_value));
   *y_offset = round_flt(((MAX_Y_VAL - y_range) / 2.0) -
(min_y * res_value));
   *z_offset = round_flt(((z_control_table
[MAX_Z_INDEX-1] - z_range) / 2.0) - (min_z *
res_value));

} /* get_offsets */




/*********************************************************
   local procedure

   check_tri_dimensions


*********************************************************/

static void check_tri_dimensions (tri_ptr)

   INPUT_TRI_STRUC *tri_ptr;

{
   if ((tri_ptr-> x1 > MAX_X_VAL) || (tri_ptr-> x1 <
0.0) ||
```

```
        (tri_ptr-> x2 > MAX_X_VAL) || (tri_ptr-> x2 <
0.0) ||
        (tri_ptr-> x3 > MAX_X_VAL) || (tri_ptr-> x3 <
0.0) ||
        (tri_ptr-> y1 > MAX_Y_VAL) || (tri_ptr-> y1 <
0.0) ||
        (tri_ptr-> y2 > MAX_Y_VAL) || (tri_ptr-> y2 <
0.0) ||
        (tri_ptr-> y3 > MAX_Y_VAL) || (tri_ptr-> y3 <
0.0) ||
        (tri_ptr-> z1 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z1 < 0.0) ||
        (tri_ptr-> z2 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z2 < 0.0) ||
        (tri_ptr-> z3 > z_control_table [MAX_Z_INDEX-1])
|| (tri_ptr-> z3 < 0.0))
      {
        fprintf (stderr, "\n error - dimensions of
uncentered object");
        fprintf (stderr, " exceed limits.  Check
Resolution value. \n");
        exit (SYS_FAILURE);
      }

} /* check_tri_dimensions */



/*******************************************************
   local procedure

   res_triangles
```

```
**********************************************************/


static void res_triangles (tri_list, x_offset, y_offset,
z_offset,
                                res_value, centering_off)

   INPUT_TRI_STRUC *tri_list;
   float           x_offset, y_offset, z_offset,
res_value;
   BOOLEAN         centering_off;

{
   INPUT_TRI_STRUC *tri_ptr;
   float           round_flt();


   tri_ptr = tri_list;

   if (centering_off)
      while (tri_ptr != NULL_INPUT_TRI)
         {
          tri_ptr-> x1 = round_flt(tri_ptr-> x1 *
res_value);
          tri_ptr-> x2 = round_flt(tri_ptr-> x2 *
res_value);
          tri_ptr-> x3 = round_flt(tri_ptr-> x3 *
res_value);

          tri_ptr-> y1 = round_flt(tri_ptr-> y1 *
res_value);
          tri_ptr-> y2 = round_flt(tri_ptr-> y2 *
res_value);
          tri_ptr-> y3 = round_flt(tri_ptr-> y3 *
res_value);
```

```
            tri_ptr-> z1 = round_flt(tri_ptr-> z1 *
        res_value);
            tri_ptr-> z2 = round_flt(tri_ptr-> z2 *
        res_value);
            tri_ptr-> z3 = round_flt(tri_ptr-> z3 *
        res_value);

            check_tri_dimensions (tri_ptr);
            tri_ptr = tri_ptr-> next_input_tri;
           }
      else
         while (tri_ptr != NULL_INPUT_TRI)
            {
            tri_ptr-> x1 = round_flt((tri_ptr-> x1 *
        res_value) + x_offset);
            tri_ptr-> x2 = round_flt((tri_ptr-> x2 *
        res_value) + x_offset);
            tri_ptr-> x3 = round_flt((tri_ptr-> x3 *
        res_value) + x_offset);

            tri_ptr-> y1 = round_flt((tri_ptr-> y1 *
        res_value) + y_offset);
            tri_ptr-> y2 = round_flt((tri_ptr-> y2 *
        res_value) + y_offset);
            tri_ptr-> y3 = round_flt((tri_ptr-> y3 *
        res_value) + y_offset);

            tri_ptr-> z1 = round_flt((tri_ptr-> z1 *
        res_value) + z_offset);
            tri_ptr-> z2 = round_flt((tri_ptr-> z2 *
        res_value) + z_offset);
            tri_ptr-> z3 = round_flt((tri_ptr-> z3 *
        res_value) + z_offset);
            tri_ptr = tri_ptr-> next_input_tri;
           }

      } /* res_triangles */
```

```
/*********************************************************
   local procedure

   remove_collapsed_triangles


*********************************************************/


static void remove_collapsed_triangles (tri_list)

   INPUT_TRI_STRUC **tri_list;

{
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
   INPUT_TRI_STRUC head_rec;
   float           Ax,Ay,Az;
   float           Bx,By,Bz;
   float           Cx,Cy,Cz;
   unsigned long   point_count = 0,line_count = 0;
   BOOLEAN         delete_tri;


   tri_ptr = *tri_list;
   head_rec .next_input_tri = tri_ptr;
   back_ptr = &head_rec;

   while (tri_ptr != NULL_INPUT_TRI) {
     delete_tri = FALSE;
     if ((tri_ptr->x1 == tri_ptr->x2) && (tri_ptr->x2 ==
tri_ptr->x3) &&
         (tri_ptr->y1 == tri_ptr->y2) && (tri_ptr->y2 ==
tri_ptr->y3) &&
```

```
          (tri_ptr->z1 == tri_ptr->z2) && (tri_ptr->z2 ==
tri_ptr->z3)) {
        delete_tri = TRUE;
        point_count++;
        }
      else {
        Ax = tri_ptr-> x2 - tri_ptr-> x1;
        Ay = tri_ptr-> y2 - tri_ptr-> y1;
        Az = tri_ptr-> z2 - tri_ptr-> z1;
        Bx = tri_ptr-> x3 - tri_ptr-> x1;
        By = tri_ptr-> y3 - tri_ptr-> y1;
        Bz = tri_ptr-> z3 - tri_ptr-> z1;
        Cx = fabs(Ay*Bz-Az*By);
        Cy = fabs(Az*Bx-Ax*Bz);
        Cz = fabs(Ax*By-Ay*Bx);
        if ((Cx < ZERO_DEF) && (Cy < ZERO_DEF) && (Cz <
ZERO_DEF)) {
          delete_tri = TRUE;
          line_count++;
          };
        };
      if (delete_tri) {
        back_ptr-> next_input_tri = tri_ptr->
next_input_tri;
        mem_free (tri_pt-.sizeof(INPUT_TRI_STRUC));
        tri_ptr = back_ptr-> next_input_tri;
        }
      else {
        tri_ptr = tri_ptr-> next_input_tri;
        back_ptr = back_ptr-> next_input_tri;
        };
       };

   *tri_list = head_rec .next_input_tri;

   if (point_count > 0) {
```

```
     fprintf (stderr, "\n warning - %lu collapsed point
triangles removed \n",
              point_count);
     fprintf (message_file, "\n warning - %lu collapsed
point triangles removed \n",
              point_count);
     };
   if (line_count > 0) {
      if (point_count == 0) {
        fprintf(stderr,"\n");
        fprintf(message_file,"\n");
        };
      fprintf (stderr, " warning - %lu collapsed line
triangles removed \n",
               line_count);
      fprintf (message_file, " warning - %lu collapsed
line triangles removed \n",
               line_count);
     };

} /* remove_collapsed_triangles */




/*********************************************************
   local procedure

   get_z_output_coord


*********************************************************/

static UINT16 get_z_output_coord (z)
```

```
   UINT16 z;

{
   UINT16 i, greater_z, lesser_z;

   i = get_z_binarily(z);
   greater_z = z_control_table [i];
   if (z == z_control_table [i])
      lesser_z = z_control_table [i];
   else
      lesser_z = z_control_table [i-1];
/* printf("get_z_output_coord:  z=%u i=%u lesser_z=%u
greater_z=%u\n",z,i,lesser_z,greater_z); */

   if ((lesser_z == 0) || (((greater_z - z) <= (z -
lesser_z)) &&
                                  (greater_z !=
z_control_table [MAX_Z_INDEX-1])))
      return (greater_z);
   else
      return (lesser_z);

} /* get_z_output_coord */



/*******************************************************
   local procedure

   adjust_z_coords



*******************************************************/
```

```
static void adjust_z_coords (tri_list, z_spacing)

    INPUT_TRI_STRUC *tri_list;
    UINT16          z_spacing;

{
    INPUT_TRI_STRUC *tri_ptr;
    float           round_flt();
    UINT16          get_z_coord();


    if (z_spacing != 1)
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = round_flt((float) tri_ptr-> z1
/ z_spacing) * z_spacing;
            tri_ptr-> z2 = round_flt((float) tri_ptr-> z2
/ z_spacing) * z_spacing;
            tri_ptr-> z3 = round_flt((float) tri_ptr-> z3
/ z_spacing) * z_spacing;
            tri_ptr = tri_ptr-> next_input_tri;
           }
       }
    else
       {
        tri_ptr = tri_list;
        while (tri_ptr != NULL_INPUT_TRI)
           {
            tri_ptr-> z1 = get_z_output_coord ((UINT16)
tri_ptr-> z1);
            tri_ptr-> z2 = get_z_output_coord ((UINT16)
tri_ptr-> z2);
            tri_ptr-> z3 = get_z_output_coord ((UINT16)
tri_ptr-> z3);
            tri_ptr = tri_ptr-> next_input_tri;
```

```
            }
        }

} /* adjust_z_coords */


/*********************************************************
   local procedure

   output_internal_triangles


*********************************************************/

static void output_internal_triangles
(user_info,tri_list)

USER_INFO_STRUC *user_info;
INPUT_TRI_STRUC **tri_list;

{
FILE *InternalFile;
INPUT_TRI_STRUC *tri_ptr;
UINT32 count,tri_count;
float zero,i,j,k;

   zero = 0;
   tri_ptr = *tri_list;
   tri_count = 0;
   while (tri_ptr != NULL_INPUT_TRI) {
      tri_count++;
      tri_ptr = tri_ptr-> next_input_tri;
   }
   if (tri_count>0) {
```

```
      fprintf(stderr, " writing internal triangles
output file ...\n");
      InternalFile = fopen(user_info->
internal_filename,"wb");

      fprintf(InternalFile,"Slice Internal Triangle
File");
      count = 28;
      while (count<80) {              /* should write
out copy of org header */
         fprintf(InternalFile," ");
         count++;
      }
      fwrite(&tri_count,4,1,InternalFile);  /* write out
# of triangle facets */
      tri_ptr = *tri_list;
      while (tri_ptr != NULL_INPUT_TRI) {
         count = 3;
         if (tri_ptr->normal_info&NORMAL_X_POSITIVE) i =
1;
         else if
(tri_ptr->normal_info&NORMAL_X_NEGATIVE) i = -1;
              else {
                i = 0;
                count--;
                };
         if (tri_ptr->normal_info&NORMAL_Y_POSITIVE) j =
1;
         else if
(tri_ptr->normal_info&NORMAL_Y_NEGATIVE) j = -1;
              else {
                j = 0;
                count--;
                };
         if (tri_ptr->normal_info&NORMAL_Z_POSITIVE) k =
1;
```

```
            else if
(tri_ptr->normal_info&NORMAL_Z_NEGATIVE) k = -1;
                  else {
                    k = 0;
                    count--;
                    };
            switch (count) {
              case 2 : i *= 0.707106781;
                       j *= 0.707106781;
                       k *= 0.707106781;
                       break;
              case 3 : i *= 0.577350269;
                       j *= 0.577350269;
                       k *= 0.577350269;
                       break;
              };
            fwrite(&i,4,1,InternalFile);
            fwrite(&j,4,1,InternalFile);
            fwrite(&k,4,1,InternalFile);

            fwrite(tri_ptr,36,1,InternalFile);   /* write
out triangle vertices */
            fwrite(&zero,2,1,InternalFile);            /*
attribute data (none) */
            tri_ptr = tri_ptr-> next_input_tri;
         }
         fclose(InternalFile);
      }
}



/*******************************************************
   local function
```

```
   flat_tri


**********************************************************/

static flat_tri (tri_ptr)

   INPUT_TRI_STRUC *tri_ptr;

{
   return
     ((tri_ptr-> z1 == tri_ptr-> z2) && (tri_ptr-> z2 ==
tri_ptr-> z3));

} /* flat_tri */



/**********************************************************
   local function

   near_flat_tri


**********************************************************/

static near_flat_tri (tri_ptr, min_angle)

   INPUT_TRI_STRUC *tri_ptr;
   INT16           min_angle;
```

```
{
   double dx1, dy1, dz1, dx2, dy2, dz2, dx, dy, dz, d,
angle;

   if (!min_angle)
      return (FALSE);
   else
      {
       dx1 = tri_ptr-> x1 - tri_ptr-> x2;
       dy1 = tri_ptr-> y1 - tri_ptr-> y2;
       dz1 = tri_ptr-> z1 - tri_ptr-> z2;

       dx2 = tri_ptr-> x1 - tri_ptr-> x3;
       dy2 = tri_ptr-> y1 - tri_ptr-> y3;
       dz2 = tri_ptr-> z1 - tri_ptr-> z3;

       dx = dy1 * dz2 - dz1 * dy2;
       dy = dz1 * dx2 - dx1 * dz2;
       dz = dx1 * dy2 - dy1 * dx2;

       d = sqrt (dx * dx + dy * dy);

       if (d == 0) angle = 0;
       else angle = 90.0 - fabs (atan (dz / d) /
RADIANS_PER_DEGREE);

       return (angle < (double) min_angle);
      }

} /* near_flat_tri */

/*************************  ***************************

  cross_product

  Computes the cross product of 2 vectors
```

```
*********************************************************/

void cross_product(a1,a2,a3,b1,b2,b3,i,j,k)
  double a1,a2,a3;  /* Point A */
  double b1,b2,b3;  /* Point B */
  double *i,*j,*k;  /* A X B */
  {
  /*
    Normal i,j,k - AxB = (a2*b3 - a3*b2)i + (a3*b1 -
a1*b3)j + (a1*b2 - a2*b1)k
  */

  *i = a2*b3 - a3*b2;
  *j = a3*b1 - a1*b3;
  *k = a1*b2 - a2*b1;
  };


/*********************************************************
   recompute_normals

   Recomputes the normals of the triangles after they
have been deformed by
    adjusting the Z coords.

   Version 252 - Only recompute X and Y

*********************************************************/

static void recompute_normals(tri_list,axis)
  INPUT_TRI_STRUC *tri_list;
  INT16           axis;
  {
  UINT8  flag;
  double  a1,a2,a3;
  double  b1,b2,b3;
```

```
  double  i,j,k;
  double  temp;

  /*
    Vector A - Vertex 2 to Vertex 3 of input
    Vector B - Vertex 2 to Vertex 1 of input

    Note: Vectors change if axis is not inputs (-x -y).
Must adjust for
          these cases.
  */

  while (tri_list != NULL_INPUT_TRI) {
    switch (axis) {
      case 0 : a1 = tri_list->x3 - tri_list->x2; /* No
axis swap */
               a2 = tri_list->y3 - tri_list->y2;
               a3 = tri_list->z3 - tri_list->z2;
               b1 = tri_list->x1 - tri_list->x2;
               b2 = tri_list->y1 - tri_list->y2;
               b3 = tri_list->z1 - tri_list->z2;
               break;
      case 1 : a3 = tri_list->x3 - tri_list->x2; /* X
and Z axis swapped */
               a2 = tri_list->y3 - tri_list->y2;
               a1 = tri_list->z3 - tri_list->z2;
               b3 = tri_list->x1 - tri_list->x2;
               b2 = tri_list->y1 - tri_list->y2;
               b1 = tri_list->z1 - tri_list->z2;
               break;
      case 2 : a1 = tri_list->x3 - tri_list->x2; /* Y
and Z axis swapped */
               a3 = tri_list->y3 - tri_list->y2;
               a2 = tri_list->z3 - tri_list->z2;
               b1 = tri_list->x1 - tri_list->x2;
               b3 = tri_list->y1 - tri_list->y2;
               b2 = tri_list->z1 - tri_list->z2;
```

```
                break;
        };

    cross_product(a1,a2,a3,b1,b2,b3,&i,&j,&k);

    switch (axis) {                              /* reswap
flag info */
      case 1: temp = k;
              k = i;
              i = temp;
              break;
      case 2: temp = k;
              k = j;
              j = temp;
              break;
      };

    flag = tri_list->normal_info &
(NORMAL_Z_POSITIVE+NORMAL_Z_NEGATIVE);

    if (fabs(i) > ZERO_DEF) {
      if (i < 0) flag |= NORMAL_X_NEGATIVE;
      else flag |= NORMAL_X_POSITIVE;
      };

    if (fabs(j) > ZERO_DEF) {
      if (j < 0) flag |= NORMAL_Y_NEGATIVE;
      else flag |= NORMAL_Y_POSITIVE;
      };

/*
    if (fabs(k) > ZERO_DEF) {
      if (k < 0) flag |= NORMAL_Z_NEGATIVE;
      else flag |= NORMAL_Z_POSITIVE;
      };
*/
```

```
    tri_list->normal_info = flag;
    tri_list = tri_list->next_input_tri;
    };
  };

/*******************************************************
   local procedure

*******************************************************/


static void get_tri_lists (in_file, user_info,
res_value, min_angle,
                           centering_off, z_spacing,
                           scan_tri_list, flat_tri_list,
near_flat_tri_list)

   FILE            *in_file;
   USER_INFO_STRUC *user_info;
   float           res_value;
   INT16           min_angle;
   BOOLEAN         centering_off;
   UINT16          z_spacing;
   INPUT_TRI_STRUC **scan_tri_list;
   INPUT_TRI_STRUC **flat_tri_list;
   INPUT_TRI_STRUC **near_flat_tri_list;

{
   INPUT_TRI_STRUC *input_tri_list, *tri_ptr, *temp_ptr;

   unsigned long   up_scan_tri_count = 0,
dn_scan_tri_count = 0;
   unsigned long   up_flat_tri_count = 0,
dn_flat_tri_count = 0;
   unsigned long   up_near_flat_tri_count = 0,
dn_near_flat_tri_count = 0;
```

```
   float           min_x, min_y, min_z, max_x, max_y,
max_z;
   float           x_offset, y_offset, z_offset;
   long            resoluted_min_z,resoluted_max_z;


   min_x = MAX_X_VAL;
   min_y = MAX_Y_VAL;
   min_z = z_control_table [MAX_Z_INDEX-1];

   max_x = 0.0 - MAX_X_VAL;
   max_y = 0.0 - MAX_Y_VAL;
   max_z = 0.0 - z_control_table [MAX_Z_INDEX-1];

/*
printf ("\n get_tri_lists - begin \n");
*/
   fprintf (stderr, "\n loading input database ...\n");

   if (user_info-> binary_input)
      pfp_get_input_tri_list_bin (in_file,
&input_tri_list, &min_x, &min_y,
                                  &min_z, &max_x,
&max_y, &max_z, user_info);
   else
      pfp_get_input_tri_list (in_file, &input_tri_list,
&min_x, &min_y,
                               &min_z, &max_x, &max_y,
&max_z,user_info);

   resoluted_min_z = round_flt(min_z * res_value);
   resoluted_max_z = round_flt(max_z * res_value);
   if (resoluted_min_z < z_control_table[0]) {
     if (*(user_info-> z_space_filename) != '\0')
       fprintf(stderr,"\nError - Part begins below
Variable Z Spacing Table\n");
     else
```

```
        fprintf(stderr,"\nError - Part begins below
minimum Z value\n");
     fprintf(stderr," Z Spacing Table: %u Part:
%ld\n",z_control_table[0],resoluted_min_z);
     exit (SYS_FAILURE);
     );
   if (resoluted_max_z > z_control_table[MAX_Z_INDEX-1])
{
     if (*(user_info-> z_space_filename) != '\0')
       fprintf(stderr,"\nError - Part extends past the
Variable Z Spacing Table of %u
layers\n",(MAX_Z_INDEX-1));
     else
       fprintf(stderr,"\nError - Part extends past the
maximum Z value of %u layers\n",(MAX_Z_INDEX-1));
     fprintf(stderr," Z Spacing Table: %u Part:
%u\n",z_control_table[MAX_Z_INDEX-1],resoluted_max_z);
     exit (SYS_FAILURE);
     );

   get_offsets (min_x, min_y, min_z, max_x, max_y,
max_z, res_value,
                &x_offset, &y_offset, &z_offset);

   res_triangles (input_tri_list, x_offset, y_offset,
z_offset,
                  res_value, centering_off);

   adjust_z_coords (input_tri_list, z_spacing);

   remove_collapsed_triangles (&input_tri_list);

   recompute_normals(input_tri_list,user_info->axis);

   if (*(user_info-> internal_filename) > '\0')
      output_internal_triangles
(user_info,&input_tri_list);
```

```
   while (input_tri_list != NULL_INPUT_TRI)
      {
       tri_ptr = input_tri_list;
       input_tri_list = input_tri_list-> next_input_tri;


       if (flat_tri (tri_ptr))
          {
           tri_ptr->normal_info = (tri_ptr->normal_info
&(255-NORMAL_X_NEGATIVE-NORMAL_X_POSITIVE
-NORMAL_Y_NEGATIVE-NORMAL_Y_POSITIVE));
           tri_ptr-> next_input_tri = *flat_tri_list;
           *flat_tri_list = tri_ptr;
           if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
              up_flat_tri_count++;
           else
              dn_flat_tri_count++;
          }
       else if (near_flat_tri (tri_ptr, min_angle))
          {
           temp_ptr = (INPUT_TRI_STRUC *) mem_alloc
(sizeof (INPUT_TRI_STRUC));
           copy (tri_ptr, temp_ptr, sizeof
(INPUT_TRI_STRUC));
           temp_ptr-> next_input_tri = *scan_tri_list;
           *scan_tri_list = temp_ptr;
           tri_ptr-> next_input_tri =
*near_flat_tri_list;
           *near_flat_tri_list = tri_ptr;
           if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
              up_near_flat_tri_count++;
           else
```

```
                dn_near_flat_tri_count++;
            }
        else
            {
             tri_ptr-> next_input_tri = *scan_tri_list;
             *scan_tri_list = tri_ptr;
             if (tri_ptr-> normal_info &
NORMAL_Z_POSITIVE)
                up_scan_tri_count++;
             else
                dn_scan_tri_count++;
            }
       }

   fprintf (stderr, "    %5lu flat triangles loaded
(%5lu up facing, %5lu down facing)\n",
                        up_flat_tri_count +
dn_flat_tri_count,
                        up_flat_tri_count,
dn_flat_tri_count);
   fprintf (stderr, "    %5lu near flat triangles loaded
(%5lu up facing, %5lu down facing)\n",
                        up_near_flat_tri_count +
dn_near_flat_tri_count,
                        up_near_flat_tri_count,
dn_near_flat_tri_count);
   fprintf (stderr, "    %5lu steep triangles loaded
(%5lu up facing, %5lu down facing)\n\n",
                        up_scan_tri_count +
dn_scan_tri_count,
                        up_scan_tri_count,
dn_scan_tri_count);



   fprintf (message_file, " %5lu flat triangles loaded
  (%5lu up facing, %5lu down facing)\n",
```

```
                        up_flat_tri_count +
dn_flat_tri_count,
                        up_flat_tri_count,
dn_flat_tri_count);
   fprintf (message_file, " %5lu near flat triangles
loaded (%5lu up facing, %5lu down facing)\n",
                        up_near_flat_tri_count +
dn_near_flat_tri_count,
                        up_near_flat_tri_count,
dn_near_flat_tri_count);
   fprintf (message_file, " %5lu steep triangles loaded
  (%5lu up facing, %5lu down facing)\n\n",
                        up_scan_tri_count +
dn_scan_tri_count,
                        up_scan_tri_count,
dn_scan_tri_count);

/*
printf ("\n get_tri_lists - end \n");
*/

} /* get_tri_lists */



/*******************************************************
   local procedure

   insert_tri



*******************************************************/


static void insert_tri (input_tri, z_bucket_array)
```

```
   INPUT_TRI_STRUC *input_tri;
   INPUT_TRI_STRUC **z_bucket_array;

{
   float          min_z;
   UINT16         min_z_value;
   UINT16         i;

   pfp_get_min_z (input_tri, &min_z);
   min_z_value = (UINT16) min_z;
   i = get_z_binarily(min_z_value);

   input_tri-> next_input_tri = z_bucket_array [i];
   z_bucket_array [i] = input_tri;

} /* insert_tri */




void tri_bucket_sort (tri_list, a_tri_bucket_array)

   INPUT_TRI_STRUC **tri_list;
   INPUT_TRI_STRUC **a_tri_bucket_array;

{
   INPUT_TRI_STRUC *tri_ptr, **bucket_ptr;
   INT16           i;

/*
printf ("\n tri_bucket_sort - begin \n");
*/
   bucket_ptr = a_tri_bucket_array;

   for (i = 0; i < MAX_Z_INDEX; i++, bucket_ptr++)
      *bucket_ptr = NULL_INPUT_TRI;
```

```
   while (*tri_list != NULL_INPUT_TRI)
      {
       tri_ptr = (*tri_list)-> next_input_tri;
       insert_tri (*tri_list, a_tri_bucket_array);
       *tri_list = tri_ptr;
      }

   *tri_list = NULL_INPUT_TRI;

/*
printf ("\n tri_bucket_sort - end \n");
*/
} /* tri_bucket_sort */



void pfp_get_input_database (in_file, user_info)

   FILE            *in_file;
   USER_INFO_STRUC *user_info;

{
   INPUT_TRI_STRUC *scan_tri_list;
   INPUT_TRI_STRUC *flat_tri_list;
   INPUT_TRI_STRUC *near_flat_tri_list;
   UINT16          z;


   scan_tri_list      = NULL_INPUT_TRI;
   flat_tri_list      = NULL_INPUT_TRI;
   near_flat_tri_list = NULL_INPUT_TRI;

   get_tri_lists (in_file, user_info,
                  user_info-> res_value, user_info->
min_surface_angle,
```

```
                    user_info-> centering_off, user_info->
z_spacing,
                    &scan_tri_list, &flat_tri_list,
&near_flat_tri_list);

  for (z = 0; z < MAX_Z_INDEX; z++)
    {
     FU_seg_bucket_array[z] = NULL_SEG;
     FD_seg_bucket_array[z] = NULL_SEG;
     NFU_seg_bucket_array[z] = NULL_SEG;
     NFD_seg_bucket_array[z] = NULL_SEG;
     }

   if (scan_tri_list != NULL_INPUT_TRI)
      tri_bucket_sort (&scan_tri_list,
tri_bucket_array);

   if (flat_tri_list != NULL_INPUT_TRI)
      process_flat_tri_list (flat_tri_list);

   if (near_flat_tri_list != NULL_INPUT_TRI)
      process_near_flat_tri_list (near_flat_tri_list);

} /* pfp_get_input_database */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPRIE.C                                          */
/*                                                   */
/* PFP Remove Internal Edges Module                  */
/*****************************************************/


#include <stdio.h>
#include <math.h>
#include "pfp.h"
#include "pfpseg.h"
#include "pfputl.h"

static OUT_SEG_STRUC *hash_table     [MAX_HASH_INDEX];
static OUT_SEG_STRUC *end_hash_table [MAX_HASH_INDEX];
static BOOLEAN       hash_table_initialized = FALSE;

/*******************************************************
   local procedure

   order_seg_points



*******************************************************/


static void order_seg_points (seg_ptr)
  OUT_SEG_STRUC *seg_ptr;
  {
  UINT16        temp;
```

```
  if ((seg_ptr-> x_1 > seg_ptr-> x_2) || ((seg_ptr->x_1
== seg_ptr->x_2) &&
                                                (seg_ptr->y_1
>  seg_ptr->y_2))) {
     temp = seg_ptr-> x_1;
     seg_ptr-> x_1 = seg_ptr-> x_2;
     seg_ptr-> x_2 = temp;
     temp = seg_ptr-> y_1;
     seg_ptr-> y_1 = seg_ptr-> y_2;
     seg_ptr-> y_2 = temp;
     };

} /* order_seg_points */




/*********************************************************
   local procedure

   insert_segment



*********************************************************/


static void insert_segment (seg_ptr)
  OUT_SEG_STRUC  *seg_ptr;
  {
  OUT_SEG_STRUC  *scan_ptr,*back_ptr;
  double         hash_key;
  UINT16         hash_index;
```

```
  hash_key = (double) MAX_Y_VAL * seg_ptr-> x_1 +
seg_ptr-> y_1;

  hash_index = (UINT16) fmod(hash_key,(double)
MAX_HASH_INDEX);

  scan_ptr = hash_table [hash_index];

  while ((scan_ptr != NULL_SEG) &&
         !((scan_ptr-> x_1 == seg_ptr-> x_1) &&
           (scan_ptr-> y_1 == seg_ptr-> y_1) &&
           (scan_ptr-> x_2 == seg_ptr-> x_2) &&
           (scan_ptr-> y_2 == seg_ptr-> y_2))) {
    back_ptr = scan_ptr;
    scan_ptr = scan_ptr-> next_seg;
    };

  if (scan_ptr == NULL_SEG) {
    if (hash_table[hash_index] == NULL)
hash_table[hash_index] = seg_ptr;
    else end_hash_table[hash_index]->next_seg = seg_ptr;

    seg_ptr->next_seg = NULL_SEG;
    end_hash_table[hash_index] = seg_ptr;
    }
  else {
    if (hash_table[hash_index] == scan_ptr)
hash_table[hash_index] = scan_ptr->next_seg;
    else {
      back_ptr->next_seg = scan_ptr->next_seg;
      if (end_hash_table[hash_index] == scan_ptr)
end_hash_table[hash_index] = back_ptr;
      };
    mem_free (scan_ptr,sizeof(OUT_SEG_STRUC));
    mem_free (seg_ptr,sizeof(OUT_SEG_STRUC));
    };
```

```
  }; /* insert_segment */


/*********************************************************
    exported procedure

    pfp_remove_duplicate_edges


*********************************************************/

void pfp_remove_duplicate_edges (seg_list)
  OUT_SEG_STRUC **seg_list;
  {
  OUT_SEG_STRUC *seg_ptr;
  UINT16        i;

  if (!hash_table_initialized) {
    for (i = 0; i < MAX_HASH_INDEX; i++) hash_table [i]
= NULL_SEG;
    hash_table_initialized = TRUE;
    };

  while (*seg_list != NULL_SEG) {
    seg_ptr = *seg_list;
    *seg_list = seg_ptr->next_seg;
    if ((seg_ptr->x_1 == seg_ptr->x_2) &&
        (seg_ptr->y_1 == seg_ptr->y_2)) {
      mem_free(seg_ptr,sizeof(OUT_SEG_STRUC));
      }
    else {
      order_seg_points (seg_ptr);
      insert_segment (seg_ptr);
      };
```

```
    );

    for (i = 0; i < MAX_HASH_INDEX; i++)
      if (hash_table[i] != NULL_SEG) {
        end_hash_table[i]->next_seg = *seg_list;
        *seg_list = hash_table[i];
        hash_table [i] = NULL_SEG;
        );

} /* pfp_remove_duplicate_edges */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPRIF.C                                          */
/*                                                   */
/* PFP Read Input File Module                        */
/*****************************************************/


#include <stdio.h>
#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"



#define NORMAL_FLAG_CHAR 'f'
#define NORMAL_MIN 1E-05

static UINT16       tri_count = 0;

/* following change made for silicon graphics
workstation only */
void convert_int(int_ptr)
  int *int_ptr;
  {
#ifdef mips
  /* if iris system , reverse 4 bytes for facet count */

  unsigned char ch,*pointer;

  pointer = (unsigned char *) int_ptr;
  ch = *pointer;
  *pointer = *(pointer+3);
```

```
  *(pointer+3) = ch;
  ch = *(pointer+1);
  *(pointer+1) = *(pointer+2);
  *(pointer+2) = ch;
#endif
  );
```

```
/*********************************************************
   local function

   is_legal_char

   Returns a value that indicates if the char parameter
is within the legal character set.

*********************************************************/

static BOOLEAN is_legal_char (ch)

   STRING ch;

{

   return ((ch >= ' ') && (ch <= '~'));
          /* the next line of code stops the '+' and '-'
illegal char messages
             but causes other problems:
               run #1:  got memory parity error, system
halted
```

```
                run #2:  got 'invalid input file format'
after several minutes
                              and program stopped, no other
information
                then commented out line and recompiled to
make it like
                  it was before  */

} /* is_legal_char */




/*********************************************************
   local procedure

   get_line

   Reads characters from the input file into a line
buffer up to the next newline char.  Checks for error.
Indicates if the file is eof.

*********************************************************/


static void get_line (input_file, line_buf, error, eof)

   FILE    *input_file;
   STRING  *line_buf;
   BOOLEAN *error;
   BOOLEAN *eof;

{
   short   status;
   BOOLEAN eoln;
```

```
   BOOLEAN char_error = FALSE;
   BOOLEAN leading_blanks_skipped = FALSE;

   do {
       do {
           status = fscanf (input_file, "%c", line_buf);

       } while (!leading_blanks_skipped && (*line_buf
== ' ') &&
                   (status == 1));
      leading_blanks_skipped = TRUE;
      *eof = (status != 1) || (*line_buf == 26);
      eoln = (*line_buf == '\n') || (*line_buf == '\r')
|| (*eof);
      if (!(eoln || *eof) && (!is_legal_char
(*line_buf)))
         {
          char_error = TRUE;
          fprintf (stderr, " Error - Illegal char found
in triangle %u: %02X (Hex)\n", tri_count,(unsigned char)
*line_buf);
         }
      line_buf++;
   } while (!*eof && !eoln);
   *--line_buf = '\0';
   *error = *error || char_error;
} /* get_line */


/*******************************************************
   local procedure

   swap_vert
```

```
*****************************************************/

void swap_vert(x,y,z,user_info)
  float *x;
  float *y;
  float *z;
  USER_INFO_STRUC *user_info;
  {
  float temp;

  if (user_info-> axis == 1) {
    temp = *z;
    *z = *x;
    *x = temp;
    }
  else if (user_info-> axis == 2) {
    temp = *z;
    *z = *y;
    *y = temp;
    }
  };

/*******************************************************
   local procedure

   get_input_tri_asc


*******************************************************/


static void get_input_tri_asc (in_file, tri_ptr,
user_info, error)

   FILE            *in_file;
   INPUT_TRI_STRUC *tri_ptr;
```

```
   USER_INFO_STRUC *user_info;
   BOOLEAN         *error;

{
   UINT16 items_read;

   tri_ptr-> next_input_tri = NULL_INPUT_TRI;

   fscanf (in_file, "%*s %f %f %f", &tri_ptr->x1,
&tri_ptr->y1, &tri_ptr->z1);

swap_vert(&tri_ptr->x1,&tri_ptr->y1,&tri_ptr->z1,user_in
fo);

   fscanf (in_file, "%*s %f %f %f", &tri_ptr->x2,
&tri_ptr->y2, &tri_ptr->z2);

swap_vert(&tri_ptr->x2,&tri_ptr->y2,&tri_ptr->z2,user_in
fo);

   items_read = fscanf (in_file, "%*s %f %f %f",
&tri_ptr->x3, &tri_ptr->y3, &tri_ptr->z3);

swap_vert(&tri_ptr->x3,&tri_ptr->y3,&tri_ptr->z3,user_in
fo);

   *error = (items_read != 3);

   if (*error)
     if (feof(in_file))
        fprintf (stderr, "\n Error - Premature end of
input file at triangle %u\n",tri_count);
     else fprintf (stderr, "\n Error -  I/O error in
reading input file at triangle %u\n",tri_count);

} /* get_input_tri_asc */
```

```
/*********************************************************
   local procedures

   update_min_max_values


*********************************************************/

void update_min_max_values (tri_ptr, min_x, min_y,
min_z,
                                                max_x,
max_y, max_z)

   INPUT_TRI_STRUC *tri_ptr;
   float           *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;

{
   if (tri_ptr-> x1 < *min_x)
      *min_x = tri_ptr-> x1;
   if (tri_ptr-> x2 < *min_x)
      *min_x = tri_ptr-> x2;
   if  tri_ptr-> x3 < *min_x)
      *min_x = tri_ptr-> x3;

   if (tri_ptr-> y1 < *min_y)
      *min_y = tri_ptr-> y1;
   if (tri_ptr-> y2 < *min_y)
      *min_y = tri_ptr-> y2;
```

```
    if (tri_ptr-> y3 < *min_y)
       *min_y = tri_ptr-> y3;

    if (tri_ptr-> z1 < *min_z)
       *min_z = tri_ptr-> z1;
    if (tri_ptr-> z2 < *min_z)
       *min_z = tri_ptr-> z2;
    if (tri_ptr-> z3 < *min_z)
       *min_z = tri_ptr-> z3;

    if (tri_ptr-> x1 > *max_x)
       *max_x = tri_ptr-> x1;
    if (tri_ptr-> x2 > *max_x)
       *max_x = tri_ptr-> x2;
    if (tri_ptr-> x3 > *max_x)
       *max_x = tri_ptr-> x3;

    if (tri_ptr-> y1 > *max_y)
       *max_y = tri_ptr-> y1;
    if (tri_ptr-> y2 > *max_y)
       *max_y = tri_ptr-> y2;
    if (tri_ptr-> y3 > *max_y)
       *max_y = tri_ptr-> y3;

    if (tri_ptr-> z1 > *max_z)
       *max_z = tri_ptr-> z1;
    if (tri_ptr-> z2 > *max_z)
       *max_z = tri_ptr-> z2;
    if (tri_ptr-> z3 > *max_z)
       *max_z = tri_ptr-> z3;

} /* update_min_max_values */

static void check_normal(i,j,k)
  float i,j,k;
  {
  float len;
```

```
  len = i*i+j*j+k*k;

  if ((len != 0) && ((len < .9) || (len > 1.1))) {
    fprintf(stderr,"\n Error reading normal for triangle
%u.\n",tri_count);
    fprintf(stderr,"  Normal %g %g %g not a unit
normal.\n",i,j,k);
    exit(SYS_FAILURE);
    };
  };


static void set_normal_info(i,j,k,normal_info)
  float i,j,k;
  UINT8 *normal_info;
  {
  if ((i > -NORMAL_MIN) && (i < NORMAL_MIN)) i = 0.0;
  if ((j > -NORMAL_MIN) && (j < NORMAL_MIN)) j = 0.0;
  if ((k > -NORMAL_MIN) && (k < NORMAL_MIN)) k = 0.0;

  *normal_info = 0;

  if (i > 0.0)       *normal_info |= NORMAL_X_POSITIVE;
  else if (i < 0.0)  *normal_info |= NORMAL_X_NEGATIVE;

  if (j > 0.0)       *normal_info |= NORMAL_Y_POSITIVE;
  else if (j < 0.0)  *normal_info |= NORMAL_Y_NEGATIVE;

  if (k > 0.0)       *normal_info |= NORMAL_Z_POSITIVE;
  else if (k < 0.0)  *normal_info |= NORMAL_Z_NEGATIVE;

  };
```

```
static void
get_normal_asc(line_buff,normal_info,user_info)
  STRING           *line_buff;
  UINT8            *normal_info;
  USER_INFO_STRUC *user_info;
  {
  INT16 items_read;
  INT16 items_expected = 3;
  float normal_i,normal_j,normal_k;

  items_read = sscanf(line_buff,"%*s %*s %f %f
%f",&normal_i,&normal_j,&normal_k);
  check_normal(normal_i,normal_j,normal_k);

  swap_vert(&normal_i,&normal_j,&normal_k,user_info);

  if (items_read != items_expected) {
    fprintf(stderr,"\n Error - I/O error trying to read
normal in triangle %u\n",tri_count);
    exit(SYS_FAILURE);
    }


set_normal_info(normal_i,normal_j,normal_k,normal_info);


  };


/*******************************************************
   exported procedure

   pfp_get_input_tri_list


*******************************************************/
```

```
void pfp_get_input_tri_list (in_file, tri_list,
                             min_x, min_y, min_z,
                             max_x, max_y, max_z,
user_info)

   FILE             *in_file;
   INPUT_TRI_STRUC **tri_list;
   float            *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;
   USER_INFO_STRUC *user_info;

{
   STRING           line_buf[200];
   BOOLEAN          error = FALSE;
   BOOLEAN          eof = FALSE;
   UINT8            normal_info;
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;


   *tri_list = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));
   tri_ptr = *tri_list;
   get_line (in_file, line_buf, &error, &eof);
   while (!eof && !error)
      {
       while (!eof && !error && (tolower(*line_buf) !=
NORMAL_FLAG_CHAR))
           ge  line (stdin, line_buf, &error, &eof);

       if (!ec  && !error)
          {
           tri_count++;

get_normal_asc(line_buf,&normal_info,user_info);
           get_line(stdin,line_buf,&error,&eof);
```

```
            get_input_tri_asc (stdin, tri_ptr, user_info,
&error);
            tri_ptr-> normal_info = normal_info;
            if (!error)
               {
                update_min_max_values (tri_ptr, min_x,
min_y, min_z,
                                                    max_x,
max_y, max_z);
                tri_ptr-> next_input_tri =
(INPUT_TRI_STRUC *) mem_alloc
                                                   (sizeof
(INPUT_TRI_STRUC));
                back_ptr = tri_ptr;
                tri_ptr = tri_ptr-> next_input_tri;
                get_line (stdin, line_buf, &error, &eof);

               }
            }
         }

   if (error)
      exit (SYS_FAILURE);

   if (tri_count > 0)
      {
       back_ptr-> next_input_tri = NULL_INPUT_TRI;
       mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
      }
   else
      {
       fprintf (stderr, "\n Error - No triangles found
in input file");
       exit (SYS_FAILURE);
      }

} /* pfp_get_input_tri_list */
```

```
/*********************************************************
   local procedure
   get_normal_bin

*********************************************************/

get_normal_bin(in_file,tri_ptr,user_info)
  FILE            *in_file;
  INPUT_TRI_STRUC *tri_ptr;
  USER_INFO_STRUC *user_info;
  {
  float normal_i,normal_j,normal_k;
  INT16 items;
  BOOLEAN errflag;

  items = 0;
  normal_i = read_float(in_file,&items);
  normal_j = read_float(in_file,&items);
  normal_k = read_float(in_file,&items);
  if (items!=3) errflag = TRUE;
  else {
    check_normal(normal_i,normal_j,normal_k);
    swap_vert(&normal_i,&normal_j,&normal_k,user_info);

set_normal_info(normal_i,normal_j,normal_k,&tri_ptr->nor
mal_info);
    errflag = (normal_i != 0.0 || normal_j != 0.0 ||
normal_k != 0.0);
    };

  return (errflag);
```

```
}/* get_normal_bin */

/*********************************************************
   local procedure

   get_input_tri_bin



*********************************************************/


void get_input_tri_bin (in_file, tri_ptr, user_info,
error)

   FILE            *in_file;
   INPUT_TRI_STRUC *tri_ptr;
   USER_INFO_STRUC *user_info;
   BOOLEAN         *error;

{
   INT16 items_read;

   tri_ptr-> next_input_tri = NULL_INPUT_TRI;

   items_read = 0;
   tri_ptr->x1 = read_float(in_file,&items_read);
   tri_ptr->y1 = read_float(in_file,&items_read);
   tri_ptr->z1 = read_float(in_file,&items_read);
   tri_ptr->x2 = read_float(in_file,&items_read);
   tri_ptr->y2 = read_float(in_file,&items_read);
   tri_ptr->z2 = read_float(in_file,&items_read);
   tri_ptr->x3 = read_float(in_file,&items_read);
   tri_ptr->y3 = read_float(in_file,&items_read);
   tri_ptr->z3 = read_float(in_file,&items_read);
   swap_vert(&tri_ptr->x1, &tri_ptr->y1, &tri_ptr->z1,
user_info);
```

```
   swap_vert(&tri_ptr->x2, &tri_ptr->y2, &tri_ptr->z2,
user_info);
   swap_vert(&tri_ptr->x3, &tri_ptr->y3, &tri_ptr->z3,
user_info);

   *error = (items_read != 9);

   if (*error) {
     if (feof(in_file))
       fprintf (stderr, "\n Error - Premature end of
input file at triangle %u\n",tri_count);
     else
       fprintf (stderr, "\n Error - I/O error reading
verticies in triangle %u invalid format in binary input
file\n",tri_count);
     exit (SYS_FAILURE);
     }

} /* get_input_tri_bin */



/*********************************************************
   exported procedure

   pfp_get_input_tri_list_bin


*********************************************************/
void pfp_get_input_tri_list_bin (in_file, tri_list,
                                 min_x, min_y, min_z,
                                 max_x, max_y, max_z,
                                 user_info)

   FILE            *in_file;
```

```
   INPUT_TRI_STRUC **tri_list;
   float           *min_x, *min_y, *min_z, *max_x,
*max_y, *max_z;
   USER_INFO_STRUC *user_info;

{
   BOOLEAN         error = FALSE;
   INPUT_TRI_STRUC *tri_ptr, *back_ptr;
   STRING          record[81];
   unsigned long   facet_records;
   UINT16          attrib_size;

   *tri_list = (INPUT_TRI_STRUC *) mem_alloc (sizeof
(INPUT_TRI_STRUC));

   tri_ptr = *tri_list;
   record[80] = '\0';
   while (!error && (fread(record,80,1,in_file) > 0)) if
(user_info->attrib) {
     fread(&facet_records,4,1,in_file);
/* the convert_int procedure will reverse the order of 4
bytes on workstation */
/* Silicon graphics    */
     convert_int(&facet_records);

     while (!error && (facet_records != 0)) {
       tri_count++;
       facet_records--;
       get_normal_bin (in_file,tri_ptr,user_info);
       get_input_tri_bin (in_file, tri_ptr, user_info,
&error);
       if (!error) {
         fread(&attrib_size,2,1,in_file);
         while (attrib_size != 0) {
           attrib_size--;
           fread(record,1,1,in_file);
           }
```

```
          update_min_max_values (tri_ptr, min_x, min_y,
min_z, max_x, max_y, max_z);
          tri_ptr-> next_input_tri = (INPUT_TRI_STRUC *)
mem_alloc (sizeof (INPUT_TRI_STRUC));
          back_ptr = tri_ptr;
          tri_ptr = tri_ptr-> next_input_tri;
          }
        }
      }
    else while (!error && get_normal_bin
(in_file,tri_ptr,user_info)) {
      tri_count++;
      get_input_tri_bin (in_file, tri_ptr, user_info,
&error);
      if (!error) {
        fread(&attrib_size,2,1,in_file);
        while (attrib_size > 0) {
          fread(record,1,1,in_file);
          attrib_size--;
          }
        update_min_max_values (tri_ptr, min_x, min_y,
min_z, max_x, max_y, max_z);
        tri_ptr-> next_input_tri = (INPUT_TRI_STRUC *)
mem_alloc (sizeof (INPUT_TRI_STRUC));
        back_ptr = tri_ptr;
        tri_ptr = tri_ptr-> next_input_tri;
        }
      }
    if (error)
      exit (SYS_FAILURE);

    if (tri_count > 0) {
      back_ptr-> next_input_tri = NULL_INPUT_TRI;
      mem_free (tri_ptr,sizeof(INPUT_TRI_STRUC));
      }
    else {
```

```
        fprintf (stderr, "\n Error - No triangles found in
    input file");
        exit (SYS_FAILURE);
        }
      } /* pfp_get_input_tri_list_bin */
```

```
/*****************************************************/
/*               3D Systems Slice Program            */
/*          Copyright (C) 1988 3D Systems Inc.       */
/*                                                   */
/* PFPSEG.C                                          */
/*                                                   */
/* PFP Segment utility module                        */
/*****************************************************/


#include <stdio.h>
#include "pfp.h"
#include "pfphi.h"
#include "pfptri.h"
#include "pfpseg.h"


/*********************************************************
   exported procedure

   Output_Error

*********************************************************/

void Output_Error()
  {
  fprintf(stderr,"\n*** I/O Error in Slice output file
***\n");
  exit(SYS_FAILURE);
  };


/*********************************************************
   exported procedure

   pfp_print_seg_list
*********************************************************/
```

```
void pfp_print_seg_list (seg_list, z, seg_file,
section_type,
                         user_info)

   OUT_SEG_STRUC    **seg_list;
   UINT16           z;
   FILE             *seg_file;
   UINT16           section_type;
   USER_INFO_STRUC *user_info;

{
   OUT_SEG_STRUC *seg_ptr,*temp_ptr;

   if (section_type == LB_CODE) {
      if (fprintf(seg_file, "%s %u\n", L_TEXT, z)!=0)
Output_Error;
      if (*seg_list != NULL_SEG)
        if (fprintf(seg_file, "%s\n", LB_TEXT)!=0)
Output_Error;
      }

   if ((*seg_list != NULL_SEG) &&
       ((*seg_list)-> normal_info == section_type)) {
     if (!user_info-> section_codes_off)
        switch (section_type)
           {
            case (LH_CODE) :
               if (fprintf(seg_file, "%s\n",
LH_TEXT)!=0) Output_Error;
               break;
            case (NFDB_CODE) :
               if (fprintf(seg_file, "%s\n",
NFDB_TEXT)!=0) Output_Error;
               break;
            case (NFDH_CODE) :
```

```
                if (fprintf(seg_file, "%s\n",
NFDH_TEXT)!=0) Output_Error;
                break;
             case (NFDF_CODE) :
                if (fprintf(seg_file, "%s\n",
NFDF_TEXT)!=0) Output_Error;
                break;
             case (NFUB_CODE):
                if (fprintf(seg_file, "%s\n",
NFUB_TEXT)!=0) Output_Error;
                break;
             case (NFUH_CODE):
                if (fprintf(seg_file, "%s\n",
NFUH_TEXT)!=0) Output_Error;
                break;
             case (NFUF_CODE) :
                if (fprintf(seg_file, "%s\n",
NFUF_TEXT)!=0) Output_Error;
                break;
             case (FUB_CODE):
                if (fprintf(seg_file, "%s\n",
FUB_TEXT)!=0) Output_Error;
                break;
             case (FUH_CODE):
                if (fprintf(seg_file, "%s\n",
FUH_TEXT)!=0) Output_Error;
                break;
             case (FUF_CODE):
                if (fprintf(seg_file, "%s\n",
FUF_TEXT)!=0) Output_Error;
                break;
             case (FDB_CODE) :
                if (fprintf(seg_file, "%s\n",
FDB_TEXT)!=0) Output_Error;
                break;
             case (FDH_CODE) :
```

```
                if (fprintf(seg_file, "%s\n",
FDH_TEXT)!=0) Output_Error;
                break;
             case (FDF_CODE) :
                if (fprintf(seg_file, "%s\n",
FDF_TEXT)!=0) Output_Error;
                break;
             }

     seg_ptr = *seg_list;

     while ((seg_ptr != NULL_SEG) &&
            (seg_ptr-> normal_info == section_type)) {
       if (fprintf(seg_file, " %u %u %u %u\n", seg_ptr->
x_1, seg_ptr-> y_1,
         seg_ptr-> x_2, seg_ptr-> y_2)!=0) Output_Error;

       temp_ptr = seg_ptr;
       seg_ptr = seg_ptr-> next_seg;
       mem_free(temp_ptr,sizeof(OUT_SEG_STRUC));
       }
     *seg_list = NULL_SEG;
     }

} /* pfp_print_seg_list */



/*****************************************************
   exported procedure

   pfp_display_seg_list
```

```
*******************************************************/


void pfp_display_seg_list (seg_list, display_mode)

   OUT_SEG_STRUC *seg_list;
   INT16         display_mode;

{
   OUT_SEG_STRUC *seg_ptr;

   set_display_mode (display_mode);

   seg_ptr = seg_list;

   while (seg_ptr != NULL_SEG)
      {
       draw_line (seg_ptr-> x_1, seg_ptr-> y_1,
seg_ptr-> x_2, seg_ptr-> y_2);
       seg_ptr = seg_ptr-> next_seg;
      }

} /* pfp_display_seg_list */




/*******************************************************
   exported procedure

   pfp_delete_seg_list
```

```
*********************************************************/


void pfp_delete_seg_list (seg_list)

   OUT_SEG_STRUC **seg_list;

{
   OUT_SEG_STRUC *next_ptr,*seg_ptr;

   seg_ptr = *seg_list;

   while (seg_ptr != NULL_SEG)
      {
       next_ptr = seg_ptr-> next_seg;
       mem_free (seg_ptr,sizeof(OUT_SEG_STRUC));
       seg_ptr = next_ptr;
      }

   *seg_list = seg_ptr;

} /* pfp_delete_seg_list */




/*********************************************************

    swap_points

*********************************************************/
```

```
void swap_points (seg_ptr)

   OUT_SEG_STRUC *seg_ptr;

{
   UINT16       temp;

   temp = seg_ptr-> x_1;
   seg_ptr-> x_1 = seg_ptr-> x_2;
   seg_ptr-> x_2 = temp;

   temp = seg_ptr-> y_1;
   seg_ptr-> y_1 = seg_ptr-> y_2;
   seg_ptr-> y_2 = temp;

} /* swap_points */




/*********************************************************
   local procedure

   get_next_seg



*********************************************************/


static void get_next_seg (seg_ptr, seg_list,
next_seg_ptr)
```

```
    OUT_SEG_STRUC *seg_ptr;
    OUT_SEG_STRUC **seg_list;
    OUT_SEG_STRUC **next_seg_ptr;

{
    OUT_SEG_STRUC *list_ptr, *back_ptr;
    OUT_SEG_STRUC head_rec;
    BOOLEAN       seg_found = FALSE;

    *next_seg_ptr = NULL_SEG;
    back_ptr = &head_rec;
    head_rec .next_seg = *seg_list;
    list_ptr = *seg_list;

    while ((list_ptr != NULL_SEG) && !seg_found)
       {
        seg_found = (((seg_ptr-> x_1 == list_ptr-> x_2)
&&
                      (seg_ptr-> y_1 == list_ptr-> y_2))
||
                     ((seg_ptr-> x_1 == list_ptr-> x_1)
&&
                      (seg_ptr-> y_1 == list_ptr->
y_1)));

        if (!seg_found)
           {
            back_ptr = list_ptr;
            list_ptr = list_ptr-> next_seg;
           }
       }

    if (seg_found)
       {
        if (!((seg_ptr-> x_1 == list_ptr-> x_2) &&
              (seg_ptr-> y_1 == list_ptr-> y_2)))
           swap_points (list_ptr);
```

```
        back_ptr-> next_seg = list_ptr-> next_seg;
        *next_seg_ptr = list_ptr;
       }

   *seg_list = head_rec .next_seg;

} /* get_next_seg */



/*********************************************************
   exported procedure

   pfp_sort_seg_list



*********************************************************/

void pfp_sort_seg_list (list_1_ptr, loop_count)

   OUT_SEG_STRUC **list_1_ptr;
   INT16         *loop_count;

{
   OUT_SEG_STRUC *list_2_ptr, *seg_ptr, *next_seg_ptr;

   seg_ptr = NULL_SEG;
   next_seg_ptr = NULL_SEG;
   list_2_ptr = NULL_SEG;
```

```
   *loop_count = 0;

   while (*list_1_ptr != NULL_SEG)
      {
       (*loop_count)++;
       seg_ptr = *list_1_ptr;
       *list_1_ptr = (*list_1_ptr)-> next_seg;
       seg_ptr-> next_seg = list_2_ptr;
       list_2_ptr = seg_ptr;
       do
          {
           get_next_seg (seg_ptr, list_1_ptr,
&next_seg_ptr);
           if (next_seg_ptr != NULL_SEG)
              {
               next_seg_ptr-> next_seg = list_2_ptr;
               list_2_ptr = next_seg_ptr;
               seg_ptr = next_seg_ptr;
              }
          } while (next_seg_ptr != NULL_SEG);
      }

   *list_1_ptr = list_2_ptr;

} /* pfp_sort_seg_list */


/*********************************************************
   procedure

   append_seg_list


*********************************************************/
```

```
void append_seg_list (list1, list2)

   OUT_SEG_STRUC **list1;
   OUT_SEG_STRUC **list2;

{
   OUT_SEG_STRUC *list1_ptr;


   if (*list1 == NULL_SEG)
      *list1 = *list2;
   else
      {
       list1_ptr = *list1;
       while (list1_ptr-> next_seg != NULL_SEG)
          list1_ptr = list1_ptr-> next_seg;
       list1_ptr-> next_seg = *list2;
      }
   *list2 = NULL_SEG;

} /* append_seg_list */
```

```
/*****************************************************/
/*              3D Systems Slice Program             */
/*         Copyright (C) 1988 3D Systems Inc.        */
/*                                                   */
/* PFPZACT.C                                         */
/*                                                   */
/* PFP Z Active list module                          */
/*****************************************************/


#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfputl.h"



/*********************************************************
   exported procedure

   pfp_update_z_active_list



*********************************************************/


void pfp_update_z_active_list (z_bucket_ptr, seg_list,
z_active_list)

   INPUT_TRI_STRUC **z_bucket_ptr;
   OUT_SEG_STRUC   **seg_list;
   VECT_TRI_STRUC  **z_active_list;
```

```
{
   VECT_TRI_STRUC  *new_vect_tri;
   INPUT_TRI_STRUC *z_bucket_back_ptr, **list_head;


   list_head = z_bucket_ptr;
   while (*z_bucket_ptr != NULL_INPUT_TRI) {
     new_vect_tri = (VECT_TRI_STRUC *) mem_alloc (sizeof
(VECT_TRI_STRUC));
     pfp_convert_tri_to_vect (*z_bucket_ptr,
new_vect_tri);
     new_vect_tri-> next_tri = *z_active_list;
     *z_active_list = new_vect_tri;
     z_bucket_back_ptr = *z_bucket_ptr;
     *z_bucket_ptr = (*z_bucket_ptr)-> next_input_tri;
     mem_free
(z_bucket_back_ptr,sizeof(INPUT_TRI_STRUC));
     );

   *list_head = NULL_INPUT_TRI;

} /* pfp_update_z_active_list */


/*********************************************************
   exported procedure

   pfp_process_z_active_list


*********************************************************/
```

```
void pfp_process_z_active_list (z_active_list, z_value,
seg_list)

   VECT_TRI_STRUC **z_active_list;
   UINT16         z_value;
   OUT_SEG_STRUC  **seg_list;

{
   VECT_TRI_STRUC *tri_back_ptr, *tri_list_ptr;
   OUT_SEG_STRUC  *seg_ptr;
   VECT_TRI_STRUC list_head;

   z_value++;

   list_head .next_tri = *z_active_list;
   tri_list_ptr = *z_active_list;
   tri_back_ptr = &list_head;

   while (tri_list_ptr != NULL_VECT_TRI)
      {
       if (tri_list_ptr-> vert3_z <= z_value)
          {
           tri_back_ptr-> next_tri = tri_list_ptr->
next_tri;
           mem_free
(tri_list_ptr,sizeof(VECT_TRI_STRUC));
           tri_list_ptr = tri_back_ptr-> next_tri;
          }
       else
          {
           seg_ptr = (OUT_SEG_STRUC *)
mem_alloc(sizeof(OUT_SEG_STRUC));
           pfp_get_intersection_segment (tri_list_ptr,
z_value, &seg_ptr);
           seg_ptr-> next_seg = *seg_list;
           *seg_list = seg_ptr;
```

```
                tri_list_ptr = tri_list_ptr-> next_tri;
                tri_back_ptr = tri_back_ptr-> next_tri;
              }
            }

          *z_active_list = list_head .next_tri;

        } /* pfp_process_z_active_list */
```

```
/*******************************************************/
/*                3D Systems Slice Program             */
/*          Copyright (C) 1988 3D Systems Inc.         */
/*                                                     */
/* PRTUTL.C - Debugging Print Routines                 */
/*                                                     */
/*                   MODIFICATION LOG                  */
/*-----------------------------------------------------*/
/* date                  who                  what     */
/*-----------------------------------------------------*/
/*             |                       |               */
/*             |                       |               */
/*******************************************************/


#include <stdio.h>
#include "pfp.h"
#include "pfptri.h"
#include "pfpseg.h"
#include "pfpchl.h"
#include "pfpcure.h"


void print_cure_hash_point(out_file,hash_point)
  FILE            *out_file;
  CURE_HASH_STRUC *hash_point;
  {
  fprintf(out_file,"Cure Hash Point %lu\n",hash_point);
  if (hash_point!=NULL_CURE_HASH) {
    fprintf(out_file,"X = %u\n",hash_point->x);
    fprintf(out_file,"Y = %u\n",hash_point->y);
    fprintf(out_file,"Segment Pointer =
%lu\n",hash_point->seg_ptr);
    fprintf(out_file,"Last Pointer    =
%lu\n",hash_point->last_hash);
    fprintf(out_file,"Next Pointer    =
%lu\n\n",hash_point->next_hash);
```

```
    }
  fflush(out_file);
  };


void print_cure_hash(out_file,hash_list,hash_index)
  FILE             *out_file;
  CURE_HASH_STRUC *hash_list;
  UINT16           hash_index;
  {
  fprintf(out_file,"\n\n\nCure Hash Index
%u:\n",hash_index);
  while (hash_list!=NULL_CURE_HASH) {
    print_cure_hash_point(out_file,hash_list);
    hash_list=hash_list->next_hash;
    };
  };


void print_hatch_point(out_file,point)
  FILE               *out_file;
  HATCH_POINT_STRUC *point;
  {
  if (point == NULL_HATCH_POINT)
fprintf(out_file,"\nHatch Point NULL\n");
  else {
    fprintf(out_file,"\nHatch Point %u
%u\n",point->x_val,point->y_val);
    fprintf(out_file,"  angle %d enter_flag %d triangle
number
%u\n",point->angle,point->enter_flag,point->triangle_num
ber);
    fprintf(out_file,"  next_point
%lu\n",point->next_point);
    };
  fflush(out_file);
```

```
  };


void print_hatch_list(out_file,point_ptr)
  FILE              *out_file;
  HATCH_POINT_STRUC *point_ptr;
  {
  fprintf(out_file,"---------- BEGIN HATCH POINT LIST
----------\n");
  while (point_ptr != (HATCH_POINT_STRUC *) NULL) {
    print_hatch_point(out_file,point_ptr);
    point_ptr = point_ptr->next_point;
    };
  fprintf(out_file,"----------- END HATCH POINT LIST
-----------\n");
  };

void print_normal_info(out_file,norm)
  FILE              *out_file;
  UINT8             norm;
  {
  fprintf (out_file, "\n");
  fprintf (out_file, "NORMAL_X_POSITIVE = %s     ",
           norm & NORMAL_X_POSITIVE ? "True " :
"False");
  fprintf (out_file, "NORMAL_X_NEGATIVE = %s\n",
           norm & NORMAL_X_NEGATIVE ? "True " :
"False");
  fprintf (out_file, "NORMAL_Y_POSITIVE = %s     ",
           norm & NORMAL_Y_POSITIVE ? "True " :
"False");
  fprintf (out_file, "NORMAL_Y_NEGATIVE = %s\n",
           norm & NORMAL_Y_NEGATIVE ? "True " :
"False");
  fprintf (out_file, "NORMAL_Z_POSITIVE = %s     ",
           norm & NORMAL_Z_POSITIVE ? "True " :
"False");
```

```
  fprintf (out_file, "NORMAL_Z_NEGATIVE = %s\n",
           norm & NORMAL_Z_NEGATIVE ? "True " :
"False");
  fprintf (out_file, "SPARE1            = %s     ",
           norm & SPARE1 ? "True " : "False");
  fprintf (out_file, "SPARE2            = %s     ",
           norm & SPARE2 ? "True " : "False");
  fprintf (out_file, "\n");
  };

void print_segment (out_file, seg)

   FILE          *out_file;
   OUT_SEG_STRUC *seg;

{
   fprintf (out_file, "\n Segment\n");
   fprintf (out_file, " x1 = %u  y1 = %u \n", seg-> x_1,
seg-> y_1);
   fprintf (out_file, " x2 = %u  y2 = %u \n", seg-> x_2,
seg-> y_2);
   print_normal_info(out_file,seg->normal_info);

   fprintf (out_file, " next ptr = %lu \n", seg->
next_seg);
   fflush(out_file);

} /* print_segment */


void print_seg_list (out_file, seg_list)

   FILE          *out_file;
   OUT_SEG_STRUC *seg_list;

{
   OUT_SEG_STRUC *seg_ptr;
```

```
    UINT16          list_length = 0;


    seg_ptr = seg_list;
    while (seg_ptr != NULL_SEG)
       {
        list_length++;
        seg_ptr = seg_ptr-> next_seg;
       }

    fprintf (out_file, "\nprinting segment list \n");
    fprintf (out_file, "list length = %u ", list_length);

    fprintf (out_file, "list =  \n");

    seg_ptr = seg_list;

    while (seg_ptr != NULL_SEG)
       {
        print_segment (out_file, seg_ptr);
        seg_ptr = seg_ptr-> next_seg;
       }
} /* print_seg_list */


void print_seg_bucket_array (out_file, seg_bucket_array)

    FILE            *out_file;
    OUT_SEG_STRUC **seg_bucket_array;

{
    OUT_SEG_STRUC *seg_ptr;
    UINT16          i;
```

```
   fprintf (out_file, "\n Z Segment Bucket
Structure:\n");

   for (i = 0; i < MAX_Z_INDEX; i++) {
       seg_ptr = seg_bucket_array [i];
       if (seg_ptr != NULL_SEG)
          fprintf (out_file, "\n\n Z Bucket Index = %d
\n", i);
       while (seg_ptr != NULL_SEG) {
           print_segment (out_file, seg_ptr);
           seg_ptr = seg_ptr-> next_seg;
          }
      }

} /* print_seg_bucket_array */




void print_input_tri (out_file, tri)

   FILE            *out_file;
   INPUT_TRI_STRUC *tri;

{
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri->
x1, tri-> y1, tri-> z1);
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri->
x2, tri-> y2, tri-> z2);
   fprintf (out_file, " (x,y,z) = %f %f %f \n", tri->
x3, tri-> y3, tri-> z3);
   print_normal_info(out_file,tri->normal_info);

} /* print_input_tri */
```

```
in_box (tri_ptr)

  INPUT_TRI_STRUC *tri_ptr;

(
   if ((tri_ptr-> z1 <= 3000) || (tri_ptr-> z2 <= 3000)
||
                                  (tri_ptr-> z3 <= 3000))

    (
     if ((tri_ptr-> z1 >= 3000) || (tri_ptr-> z2 >=
3000) ||
                                    (tri_ptr-> z3 >=
3000))
      (
       if (
           ((tri_ptr-> x1 < 20000) && (tri_ptr-> x1 >
13000)) ||
           ((tri_ptr-> x2 < 20000) && (tri_ptr-> x2 >
13000)) ||
           ((tri_ptr-> x3 < 20000) && (tri_ptr-> x3 >
13000))
         )
        (
         if (
             ((tri_ptr-> y1 < 55000) && (tri_ptr-> y1 >
48000)) ||
             ((tri_ptr-> y2 < 55000) && (tri_ptr-> y2 >
48000)) ||
             ((tri_ptr-> y3 < 55000) && (tri_ptr-> y3 >
48000))
            )
           return (1);
        )
      )
```

```
    }
  return (0);

} /* in_box */


void print_tri_list (out_file,tri_list)

  FILE            *out_file;
  INPUT_TRI_STRUC *tri_list;

{
   INPUT_TRI_STRUC *tri_ptr;
   UINT16           list_length = 0;


   tri_ptr = tri_list;
   while (tri_ptr != NULL_INPUT_TRI)
      {
       list_length++;
       tri_ptr = tri_ptr-> next_input_tri;
      }

    fprintf (out_file, "\n list length = %u ",
list_length);
    fprintf (out_file, "\n list =  \n");

   tri_ptr = tri_list;
   while (tri_ptr != NULL_INPUT_TRI) {
/*if (in_box (tri_ptr)) */
     print_input_tri (out_file, tri_ptr);
     tri_ptr = tri_ptr-> next_input_tri;
     }

} /* print_tri_list */
```

```
void print_tri_bucket_array (out_file, z_bucket_array)

   FILE             *out_file;
   INPUT_TRI_STRUC **z_bucket_array;

{
   INPUT_TRI_STRUC *tri_ptr;
   INT16            i;


   fprintf (out_file, "\n Tri Bucket Structure:\n");

   for (i = 0; i < MAX_Z_INDEX; i++)
      {
       tri_ptr = z_bucket_array [i];
       if (tri_ptr != NULL_INPUT_TRI)
          fprintf (out_file, "\n\n Z Bucket Index = %d
\n", i);
       while (tri_ptr != NULL_INPUT_TRI)
          {
           print_input_tri (out_file, tri_ptr);
           tri_ptr = tri_ptr-> next_input_tri;
          }
      }

} /* print_tri_bucket_array */



void print_vect_tri (out_file, vect_tri)

   FILE             *out_file;
```

```
    VECT_TRI_STRUC *vect_tri;

{
    fprintf (out_file, " vector 1:\n");
    fprintf (out_file, "    x = %f   y = %f   x_incr = %f
 y_incr = %f\n",
                        vect_tri-> vector_1 .x,
                        vect_tri-> vector_1 .y,
                        vect_tri-> vector_1 .x_incr,
                        vect_tri-> vector_1 .y_incr);

    fprintf (out_file, " vector 2:\n");
    fprintf (out_file, "    x = %f   y = %f   x_incr = %f
 y_incr = %f\n",
                        vect_tri-> vector_2 .x,
                        vect_tri-> vector_2 .y,
                        vect_tri-> vector_2 .x_incr,
                        vect_tri-> vector_2 .y_incr);

    fprintf (out_file, " vector 3:\n");
    fprintf (out_file, "    x = %f   y = %f   x_incr = %f
 y_incr = %f\n",
                        vect_tri-> vector_3 .x,
                        vect_tri-> vector_3 .y,
                        vect_tri-> vector_3 .x_incr,
                        vect_tri-> vector_3 .y_incr);

    fprintf (out_file, "Vertex 1 Z =
%f\n",vect_tri->vert1_z);
    fprintf (out_file, "Vertex 2 Z =
%f\n",vect_tri->vert2_z);
    fprintf (out_file, "Vertex 3 Z =
%f\n",vect_tri->vert3_z);
    print_normal_info(out_file,vect_tri->normal_info);

} /* print_vect_tri */
```

```
void print_z_active_list (out_file, z_active_list)

    FILE           *out_file;
    VECT_TRI_STRUC *z_active_list;

{
    while (z_active_list != NULL_VECT_TRI)
        {
         print_vect_tri (out_file, z_active_list);
         z_active_list = z_active_list-> next_tri;
        }
    fflush (out_file);

} /* print_z_active_list */
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================

   RECOAT1.PAS
   SLA-1 Recoater wiper and plunger Control Program
---------------------------------------------------------

   Recent History:
   11/12/87  Ver 3.30  first released for breadboard
testing
---------------------------------------------------------}
{$M 16384,0,0}

uses Crt,Dos;

const
  VersionId = '3.30';
  LaserBoardAddress = $300;
  PowerPort      = $304;
  _INIT_OUT      = $01;  { outputs }
  _ZSTAGE_OUT    = $02;
  _SHUTTER_OUT   = $04;
  _LASER_OUT     = $08;
  _INIT_IN       = $10;  { inputs }
  _DOOR_IN       = $20;
  _SHUTTER_IN    = $40;
  _LASER_IN      = $80;
  Indent = 22;
  LASERVECTOR  =  64;
  STEREOVECTOR =   5;
  HomeDirectory =  C:\3DSYS\';
  FixFileName    = 'FIX.DAT';
  ParamFileName = 'POWER.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  CtrlC = ^C;
```

```
  Zel: array[1..12] of Integer =
(1,4,3,6,1,4,6,2,5,0,3,5);
  DazeType: array[0..6] of String =
('Sunday','Monday','Tuesday','Wednesday','Thursday','Fri
day','Saturday');
  DayOfWeek : String[21] = 'SunMonTueWedThuFriSat';

type
  str4 = string[4];
  str16 = string[16];
  str10 = string[10];
  str80 = string[80];
  LaserPowerType = (_Unknown,_Level,_Toggle);
  OnOffType = (On,Off);

var
  delaytime : integer;
  key: Char;
  regs: Registers;
  ParamFile,ErrorFile,FixFile : text;
  LaserVersion,StereoVersion: str4;
  CurrentDate, CurrentTime : str80;
  temp,IndentStr,FixFileLine,ParamFileLine: String;
  StepIter,i,IOerr: Integer;
  sec,minu,hour,day,mon,year,weekday : Integer;
  QSize  : LongInt;
  LaserPower: LaserPowerType;
  LaserOff,ShowStuff: Boolean;
  SweepStep,PlungeStep : integer;
{
   Cursor on / off routines
}

function HexByte(num:Byte):str4;
const
   HexChars: str16= '0123456789ABCDEF';
begin
```

```
   HexByte := HexChars[num shr 4 + 1] + HexChars[num and
$0F +1];
end;

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;

{
  Show error and store it on disk
```

```
}
procedure SystemError(Text: string);
begin
  writeln('ERROR: '+Text);
  beep;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'POWER: ',Text);
  close(ErrorFile);
end;

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
begin
  if whereX>1 then writeln;
  writeln('*** ',Text);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
```

```
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);

  IntStr := temp;
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = CtrlC then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;
```

```
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                    {  Type         1    2    3    4
}
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
```

```
    end;
    gotoxy(x1,y2);
    if x1<>x2 then write(LowerLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(LowerRightCorner[BoxType]);
  end;

  procedure GetSelection;
  begin
    repeat
      HandleKeyboardRequest;
      key := upcase(key);
      if key in ['1'..'7','X'] then exit;
    until false;
  end;

  procedure ShowPowerMenu;
  begin
    window(1,1,80,25);
    ClrScr;
    centerln('RECOAT1, SLA-1 Recoater Utility Control
  Program');
    centerln('Version '+VersionId);
    centerln('Copyright (C) 1988 by 3D Systems, Inc.');
    centerln('3D Systems Laser Stereolithography System');

    for i:=1 to 5 do writeln;
    DrawBox(Indent-1,8,80-Indent,17,1);
    window(Indent,9,80-Indent-1,16);
    ClrScr;
    writeln(' 1. Initialize Sweeper and Plunger ');
    writeln(' 2. Move Sweeper Up');
    writeln(' 3. Move Sweeper Down');
    writeln(' 4. Move Plunger Up');
    writeln(' 5. Move Plunger Down');
    writeln(' 6. Set a new delay time');
    writeln;
```

```
  write  (' X. Exit');
  window(1,1,80,25);
  gotoxy(28,19);
  write('Please make a selection.');
  DrawBox(10,23,69,25,1);
  gotoxy(15,24);
  write('Delay between steps = ',delaytime,'
milliseconds          ');
end;

procedure UserWait;
begin
   writeln;
   writeln;
   writeln('                ... press any key to
continue ');
   repeat delay(100) until KeyPressed;
   key:=Readkey;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
```

```
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

function Zeller(month,day,year:integer):Integer;
begin
   if month<3 then dec(year);
   Zeller := ( Zel[month] + day + year + (year shr 2))
mod 7;
end;

procedure SetLaserOnOffTime;
  var
     ChronoFile  :  text;
     ChronoStr,ServiceStr   :  String;
     AbsDate     :  array[1..8] of char;
     AbsTime     :  array[1..8] of char;
     OffsetHour, OffsetMonth  :  array[1..2] of char;
     yearHold,nnnnstr    :  string[4];
     OffsetDay   :  array[1..3] of char;
     Fileexists, AM, fileempty, endoffile  : boolean;
     OnEntryExists, OffEntryExists : boolean;
     TryToAddOn, TryToAddOff       : boolean;
     j, nnn, valcode         :  integer;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetChipTime;
var
  good: Boolean;
begin
```

```
  good := false;
  while not good do begin
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;   { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;    { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;    { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;    { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $09;    { get year }
                Year := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then good :=
true;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
```

```
  Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
  WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetChipTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetChipTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
{  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear; }
  CurrentDate := DazeType[Zeller(Mon,Day,Year)]+'
'+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
```

```
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;


procedure ShowCurrentLDT;
begin
     ClrScr;
     write('      Current Laser Status = ');
     if port[PowerPort + 2] and _LASER_IN   = 0 then
writeln('On ')
     else writeln('Off');
     GetChipTimeStamp;
     writeln('      Current Date         =
',CurrentDate);
     writeln('      Current Time         =
',CurrentTime);
     writeln;
end;

function DayOfWeekZeller:integer;
var
   imonth, iday, iyear : integer;
   cmonth, cday, cyear : array[1..2] of char;
begin
   for i:= 1 to 2 do begin
      cmonth[i] := AbsDate[i];
      cday[i]   := AbsDate[i+3];
      cyear[i]  := AbsDate[i+6];
   end;
   val(cmonth,imonth,valcode);
   val(cday,iday,valcode);
   val(cyear,iyear,valcode);
```

```
   DayOfWeekZeller:= Zeller(imonth,iday,iyear);
end;


procedure IssueTimeCommand(laserCMD: str4);
   var
      CMDnoperiod : str4;
      iend        : integer;
begin
    if laserCMD = 'off.' then TryToAddOff:=true
                         else TryToAddOn :=true;
    CMDnoperiod:=laserCMD;
    CMDnoperiod[pos('.',laserCMD)]:=' ';
    ServiceStr:='/C SERVICE LASER ';
    ShowCurrentLDT;
    writeln;
    writeln('Enter:');
    writeln;
    writeln('      M  specify minutes from now for laser
',CMDnoperiod);
    writeln;
    writeln('      T  specify time and date for laser
',CMDnoperiod);
    repeat key:=upcase(Readkey) until key in ['T','M'];
    ShowCurrentLDT;
    if key='T' then begin
       writeln;
       writeln('Enter date and time to turn the laser
',laserCMD);
       writeln;
       writeln('FORMAT: mo/dd/yr hr:mn xx');
       writeln;
       writeln('  where mo=month  dd=day     yr=year');
       writeln('        hr=hour   mn=minute  xx=AM or PM
');
       writeln;
       CursorOn;
```

```
        Readln(chronoStr);
        iend:=0;
        iend:= pos('M',chronoStr);
        if iend=0 then iend:=pos('m',chronoStr);
        if iend=0 then begin
           ClrScr;
           gotoxy(1,3);
           writeln('   Invalid Entry: please specify AM
or PM. ');
           userwait;
           exit;
        end;
        if laserCMD='on. ' then ServiceStr := ServiceStr
+ 'ON DATE '
                           else ServiceStr := ServiceStr
+ 'OFF DATE ';
        for i:= 1 to pos(' ',chronoStr)
           do ServiceStr := ServiceStr+chronoStr[i];
        ServiceStr:= ServiceStr + 'TIME';
        for i:= pos(' ',chronoStr) to iend
           do ServiceStr := ServiceStr+chronoStr[i];
     end else begin
        writeln;
        writeln('Enter minutes from now when the laser
will turn ',laserCMD);
        writeln;
        writeln;
        writeln;
        CursorOn;
        Readln(chronoStr);
        val(chronoStr,nnn,valcode);
        if (valcode=0) and (nnn>0) and (nnn<999) then
begin
           Str(nnn:4,nnnnstr);
           for i:= 1 to 4 do if nnnnStr[i]=' ' then
nnnnStr[i]:='0';
```

```
          if laserCMD='on. ' then ServiceStr :=
ServiceStr + 'ON +'
                                else ServiceStr :=
ServiceStr + 'OFF +';
          ServiceStr := ServiceStr + nnnnStr;
       end;
    end;
    CursorOff;
    ServiceStr:=ServiceStr + '/i/t/P';  { priority
service }
    exec('C:\COMMAND.COM',ServiceStr);
    delay(100);
end;  { procedure IssueTimeCommand }

begin
TryToAddOn := false;
TryToAddOff:= false;
repeat
  window(1,1,80,25);
  ClrScr;
  centerln('RECOAT1, SLA-1 Recoater Utility Control
Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');

  writeln;
  centerln('Laser On/Off Timer Menu');
  DrawBox(10,8,80-10,23,1);
  window(13,9,80-15,22);
  ClrScr;
  {$I-}
  assign(chronofile,'\Chrono.sys');
  reset(chronofile);
  fileexists := (IOResult = 0);
  if not fileexists then begin
     gotoxy(1,6);
```

```
      writeln('*** ERROR : CHRONO.SYS file not found.');
      UserWait;
      exit;
   end else begin
      ShowCurrentLDT;
      chronoStr:='';
      fileempty := true;
      endoffile := false;
      OnEntryExists := false;
      OffEntryExists := false;
      for j:=1 to 2 do begin
      if not endoffile then begin
      repeat
         Readln(chronofile,ChronoStr);
         endoffile:=EOF(chronofile);
      until endoffile or ((chronostr[1]='A') and
(chronostr[3]='L'));
      if (not endoffile or (chronostr[1]='A')) then
fileempty:=false;
      if (not fileempty) and (chronostr[1]='A') then
begin
         if chronoStr[4]='O' then begin
            write('The laser is set to turn on ');
            OnEntryExists:=true
         end else begin
            write('The laser is set to turn off ');
            OffEntryExists:=true
         end;
         for i:= 1 to 8 do begin
            AbsDate[i]:=chronoStr[i+5];
            AbsTime[i]:=chronoStr[i+14];
         end;
         for i:= 1 to 2 do begin
            OffsetDay[i]:=chronoStr[i+23];
            OffsetHour[i]:=chronoStr[i+27];
            OffsetMonth[i]:=chronoStr[i+30];
         end;
```

```
        OffsetDay[3]:=chronoStr[26];
        writeln('at ');
        if (OffsetDay<>'000') or (OffsetHour<>'00') or
(OffsetMonth<>'00')
        then begin
           write('      ');
           if OffsetMonth<>'00' then write(OffsetMonth,'
month(s) ');
           if OffsetHour <>'00' then write(OffsetHour,'
hour(s) ');
           if OffsetDay <>'000' then write(OffsetDay,'
day(s) ');
           writeln('from ');
        end;
        writeln('                ',AbsTime,' on
',DazeType[DayOfWeekZeller],', ',AbsDate,'. ');
        writeln;
     end; {repeat}
     end; {if not end of file}
     end; {for j:= 1 to 2 }
  end;
  close(chronofile);
  {$I+}
  if not TryToAddOn and not TryToAddOff
  and not OnEntryExists and not OffEntryExists then
begin
     writeln;
     writeln('      No entries found in the laser timer
queue.');
  end;
  if TryToAddOn and not OnEntryExists
  then writeln('Invalid ON entry ignored.');
  if TryToAddOff and not OffEntryExists
  then writeln('Invalid OFF entry ignored.');
  gotoxy(1,12);
  writeln('Select:  1. Remove ON entry      3. Add ON
entry ');
```

```
  writeln('          2. Remove OFF entry      4. Add OFF
entry');
  write  ('              ( X to exit )
  ');
  repeat key:=upcase(readkey) until key in
['1'..'4','X'];
  if key='X' then exit;
  ClrScr;
  TryToAddOn:=false;
  TryToAddOff:=false;
  if key='1' then exec('C:\COMMAND.COM','/C SERVICE
LASER ON CANCEL/i/t');
  if key='2' then exec('C:\COMMAND.COM','/C SERVICE
LASER OFF CANCEL/i/t');
  if key='3' then if OnEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser ON entry already
exists.  ');
     UserWait;
  end else IssueTimeCommand('on. ');
  if key='4' then if OffEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser OFF entry already
exists. ');
     UserWait;
  end else IssueTimeCommand('off.');
until false;
end;

procedure HardReset;
begin
  port[PowerPort + 3] := $98;   { init 8255 chip }
  port[PowerPort + 3] := $98;
end;

procedure ReadParams;
begin
```

```
  assign(ParamFile,ParamFileName);
  {$I-}
  reset(ParamFile);
  IOerr := IOresult;
  if IOerr <> 0 then begin
    assign(ParamFile,HomeDirectory + ParamFileName);
    reset(ParamFile);
    IOerr := IOresult;
  end;
  {$I+}
  LaserPower := _Unknown;
  if IOerr = 0 then begin
    while not eof(ParamFile) do begin
      readln(ParamFile,ParamFileLine);
      ParamFileLine := upper(ParamFileLine);
      if (ParamFileLine <> '') and (ParamFileLine[1] <>
'!') then begin
        if copy(ParamFileLine,1,15) = 'LASERPOWERTYPE='
then begin
          temp := copy(ParamFileLine,16,3);
          if temp = 'LEV' then LaserPower := _Level
          else if temp = 'TOG' then LaserPower :=
_Toggle;
        end;
      end;
    end;
    close(ParamFile);
  end;
end;

procedure RotateStep(DirStep,InputByte: Integer);
const
  Bits: array[0..3] of Byte = ($01,$03,$02,$00);
begin
 StepIter := ((StepIter + DirStep) + 4) mod 4;
 InputByte := Bits[StepIter];
{}{ writeln('   InputByte =',InputByte); }
```

```
end;

procedure InitSweeperandPlunger;
begin
   port[$30B] := $98;
   port[$30B] := $98;
   gotoxy(15,24);
   write('SWEEPER AND PLUNGER INITIALIZED
   ');
   repeat delay(100) until keypressed;
   key := readkey;
end;

procedure SweeperUp;
begin
   gotoxy(15,24);
   write('SWEEPER UP - press any key to halt motion
   ');
   repeat
      rotatestep(+1,Sweepstep);
      port[$30A] := (port[$30A] and $F3) or (sweepstep*4
and $0C);
      delay(delaytime);
   until keypressed;
   key := Readkey;
end;

procedure SweeperDown;
begin
   gotoxy(15,24);
   write('SWEEPER DOWN - press any key to halt motion
     ');
   repeat
      rotatestep(-1,sweepstep);
      port[$30A] := (port[$30A] and $F3) or (sweepstep*4
and $0C);
      delay(delaytime);
```

```
   until keypressed;
   key := Readkey;
end;

procedure PlungerUp;
begin
   gotoxy(15,24);
   write('PLUNGER UP - press any key to halt motion
   ');
   repeat
      rotatestep(+1,plungestep);
      port[$30A] := (port[$30A] and $FC) or (plungestep
and $03);
      delay(delaytime);
   until keypressed;
   key := Readkey;
end;

procedure PlungerDown;
begin
   gotoxy(15,24);
   write('PLUNGER DOWN - press any key to halt motion
     ');
   repeat
      rotatestep(-1,plungestep);
      port[$30A] :=( port[$30A] and $FC) or (plungestep
and $03);
      delay(delaytime);
   until keypressed;
   key := Readkey;
end;

procedure SetNewDelay;
var
   instring : string;
   valcode  : integer;
begin
```

```
    gotoxy(15,24);
    write('
');
    gotoxy(15,24);
    write('ENTER NEW DELAY (milliseconds): ');
    cursoron;
    readln(instring);
    val(instring,delaytime,valcode);
    cursoroff;
end;

begin
  delaytime  := 10;
  CheckBreak := false;
  ShowStuff  := false;
  LaserOff   := false;
  SweepStep := $00;
  PlungeStep := $00;
  StepIter := 0;
  ClrScr;
  IndentStr := many(' ',Indent);
{  ReadParams; }
  CursorOff;
  repeat
    ShowPowerMenu;
    GetSelection;
    gotoxy(1,22);
    ClrEol;
    case key of
{     '1': TurnOnLaser;
      '2': TurnOnElevator;
      '3': OpenShutter;
      '4': TurnOffLaser;
      '5': TurnOffElevator;
      '6': CloseShutter;
      '7': SetLaserOnOffTime;   }
      '1': InitSweeperandPlunger;
```

```
            '2': SweeperUp;
            '3': SweeperDown;
            '4': PlungerUp;
            '5': PlungerDown;
            '6': SetNewDelay;
            'X': Abort;
          end;
          key := ' ';
        until false;
        CursorOn;

      end.
      ^Z
```

```
{
============================================================


    CLIFF, Control Laser Instrument From File

    for 3D Systems' Sterolithography System

    Version 2.80  by Tarnz Technologies


============================================================

 CLIFF reads simple (X,Y) laser position commands from a
file and controls the laser mirrors using the LASER
background process.  Please see the MRCLIFF program for
details on the commands and their arguments.

The command line for cliff takes the form:

      CLIFF <datafile> [/options]

where <datafile> is the name of the data file containing
the laser control information, and /options are any
number of user-selectable options:

      /S    -  show laser control information
      /W    -  wait for laser idle at end
      /T    -  transparency mode (no CLIFF text)

CLIFF can perform the wait-for-idle function (/W option)
without having any data file or commands to execute.
CLIFF /W and CLIFF /W/T (transparent) will make CLIFF
wait for the background task to idle.

<datafile> can be replaced by a sequence of direct
commands by preceding the commands with a ', ", or @
```

character. A semicolon can be used to separate commands. As many commands as will fit on one line can be specified. In the following example, the block mode is turned on and the laser beam moved:

```
CLIFF 'MD BL +; JX 1000; JY 2000; NX 4000; NY 5000; EC
```

If CLIFF is called up with no arguments, it will prompt with

```
      Enter file name: _
```

at which point the operator should enter the name of the data file containing CLIFF-compatible commands. Direct commands can be entered here if preceded by a special character as described above.

```
==========================================================
}

{

  Recent History:
   8/18/87  Ver 2.02  CLIFF Version 2.00 released
                      interfaces with memory resident
                      STEREO device driver
   8/22/87  Ver 2.03  /NVC changed to /NC for no version
                      check
   8/25/87  Ver 2.04  only updated version number
   9/23/87  Ver 2.12  added MinSystemVersion check
   9/30/87  Ver 2.25  only updated version number
  11/12/87  Ver 2.30  first Beta software release
  12/09/87  Ver 2.40  upgraded to Turbo Pascal Version
4.0
                      different STEREO-installed check
   1/25/88  Ver 2.60  version update only
```

```
   3/09/88  Ver 2.80  added copyright notice
   4/09/88  Ver 3.00  production release
   7/18/88  Ver 3.20  update version
}

uses Dos,Crt;

type
  str16 = string[16];
  str80 = string[80];
  str255 = string[255];

var
  ch,key: Char;
  Line,temp,cmd: str255;
  StereoVersion: str16;
  count,i,remline,Er,OpNum,code: Integer;
  Format,LineCount,IOerr: Integer;
  ErrorFile,InputFile: text;
  VersionCheckFlag,TransparentFlag,WaitForIdleFlag:
Boolean;
  ShowData,ErrorOpen,Abort: Boolean;
  regs: Registers;

const
  ThisCliffVersion = '3.20';
  MinSystemVersion = '3.20';
  CtrlC = #3;
  sp40 = '
  _LASERVECTOR  = $64;
  _STEREOVECTOR = $65;

label ExitProgram;

function upper(text: str255): str255;
  var
    i: Integer;
```

```
    temp: str255;
  begin
    temp := text;
    for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
    upper := temp;
  end;

function many(Ch: Char;num: Integer): str255;
  var
    temp: str255;
    i: Integer;
  begin
    temp := '';
    for i:=1 to num do temp  := temp+Ch;
    many := temp;
  end;

procedure ReadCommandLine(var line: str255);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

procedure print(text: str255);
  begin
    if not TransparentFlag then write(text);
  end;

procedure println(text: str255);
  begin
    if not TransparentFlag then writeln(text);
  end;
```

```
function IntStr(num: Integer): str16;
  var
    temp: str16;
  begin
    if num>=0 then Str(num,temp) else
Str(num+$8000,temp);
    IntStr := temp;
  end;

function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    __LaserOk := (CopyStr = 'Tarnz Technologies');
  end;

function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;

function __StereoOk: Boolean;
  type
```

```
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or _STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      __StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;

function __StereoPresent: Boolean;
  begin
    if __StereoOk then __StereoPresent := true
    else begin
      __StereoPresent := false;
      writeln('STEREO driver not installed.');
    end;
  end;

procedure Error(text: str255);
  begin
    if whereX > 1 then println('');
    if LineCount=0 then println('*** '+text)
```

```
      else println('*** '+text+' in line
'+IntStr(LineCount));
    if not ErrorOpen then begin
      rewrite(ErrorFile);
      ErrorOpen := true;
    end;
    writeln(ErrorFile,text);
  end;

procedure DescribeError(Er: Integer);
  begin
    case Er of
        0: ;
        1: Error('1. Laser interface card not found');
        2: Error('2. Laser channel not open');
        3: Error('3. Background Command Queue full
timeout');
        4: Error('4. Invalid interrupt time');
        5: Error('5. Not in block mode');
        6: Error('6. Exec Queue limit timeout');
      127: Error('127. Bad SIH command parameter');
      128: Error('128. Unrecognized STEREO command');
      255: Error('255. Internal error (please report)');

      else Error(IntStr(Er)+'. Unrecognized SIH error');

    end;
  end;

{ - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-

    Pass Command on to Memory Resident Portion

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
}
```

```
procedure StereoCommand(line: str255);
  begin
    if (length(line)>0) and (line[1]<>'!') then with
regs do begin
      line := line + chr(0);  { end-of-line }
      AX := $00;              { for future expansion }
      BX := Ofs(line)+1;      { pass pointer to string }
      ES := Seg(line);
      Intr(_STEREOVECTOR,regs);
    end;
    if regs.AL>0 then DescribeError(regs.AL);
  end;

procedure ProcessFile;
  begin
    if not ShowData then begin
      println('');
      print('0    lines');
    end;
    while not eof(InputFile) and not Abort do begin
      readln(InputFile,Line);
      if ShowData then println(Line);
      LineCount := LineCount+1;
      StereoCommand(Line);
      if not TransparentFlag then
        if not ShowData then if LineCount mod 25=0 then
begin
          gotoxy(1,whereY);
          print(IntStr(LineCount));
        end;
      if keypressed then begin
        key := ReadKey;
        if key in [CtrlC,'A','a'] then Abort := true;
      end;
    end; {not eof}
    if not (TransparentFlag and ShowData) then begin
      gotoxy(1,whereY);
```

```
      println(IntStr(LineCount));
    end;
    if Abort then begin
      Error('Control Process Aborted');
      while keypressed do key := ReadKey;
    end; {if Abort}
  end;

procedure CheckForOptions(var str: str255);
  var
    i: Integer;
  begin
    while (length(str)>1) and (str[1]=' ') do
delete(str,1,1);
    i := pos('/S',upper(str));  { show input file lines
}
    if i>0 then begin
      ShowData := true;
      delete(str,i,2);
    end;
    i := pos('/W',upper(str));  { set wait for idle flag
}
    if i>0 then begin
      WaitForIdleFlag := true;
      delete(str,i,2);
    end;
    i := pos('/T',upper(str));  { set transparency flag
}
    if i>0 then begin
      TransparentFlag := true;
      delete(str,i,2);
    end;
    i := pos('/NC',upper(str));  { no version check }
    if i>0 then begin
      VersionCheckFlag := false;
      delete(str,i,3);
    end;
```

```
  end;

procedure CliffHeader;
  begin
    writeln;
    writeln(#213,many(#205,53),#184);
    writeln(#179,'  CLIFF, Control Laser Instrument From
File Program  ',#179);
    writeln(#179,'  Version ',ThisCliffVersion+'
                            ',#179);
    writeln(#179,'  Copyright (C) 1988 by Tarnz
Technologies           ',#179);
    writeln(#179,'  3D Systems Laser Sterolithography
System          ',#179);
    writeln(#212,many(#205,53),#190);
    writeln;
  end;

begin
  ReadCommandLine(cmd);
  LineCount        := 0;
  Abort            := false;
  WaitForIdleFlag  := false;
  TransparentFlag  := false;
  ErrorOpen        := false;
  ShowData         := false;
  VersionCheckFlag := true;

{ ***** check for memory resident STEREO device driver }


  Assign(ErrorFile,'CLIFF.ERR');
  {$I-}  { erase error file if it exists }
  Erase(ErrorFile);
  {$I+}
  IOerr := IOresult;  { clear I/O result }
  CheckForOptions(cmd);
```

```
  if not TransparentFlag then CliffHeader;
  if not (__LaserPresent and __StereoPresent) then goto
ExitProgram;
  if VersionCheckFlag then
    if StereoVersion < MinSystemVersion then begin
      writeln('Obsolete STEREO in use.');
      goto ExitProgram;
    end;
  if length(cmd)=0 then begin
    write('Enter file name: ');
    readln(cmd);
    CheckForOptions(cmd);
  end;
  if (length(cmd)>1) and (cmd[1] in ['@','"',''''']) then
begin
    { handle direct commands }
    cmd := copy(cmd,2,199);
    StereoCommand(cmd);
  end else begin
    { handle commands from a file }
    if length(cmd)>0 then begin
      if pos('.',cmd)=0 then cmd := cmd + '.DAT';
      assign(InputFile,cmd);
      {$I-}
      reset(InputFile);
      IOerr := IOresult;
      {$I+}
      if IOerr <> 0 then
        Error('I/O Error '+IntStr(IOerr)+' opening file
'+cmd)
      else begin
        ProcessFile;
        close(InputFile);
      end;
    end;
  end;
```

```
  if WaitForIdleFlag then StereoCommand('WI 1000');

ExitProgram:
  {$I-}
  close(InputFile);
  close(ErrorFile);
  {$I+}

end.
^Z
```

3.20 SOURCE .PAS FILES

November 07, 1988

recoat1.pas
cliff.pas
laser.pas
monitor.pas
view.pas
beam.pas
edit.pas
libnc.pas
power.pas
xfer.pas
bench.pas
gobuild.pas
material.pas
service.pas
zstage.pas
build.pas
gostereo.pas
menu.pas
stereo.pas
recoato.pas
calout.pas
install.pas
merge.pas
testpart.pas

```
{////////////////////////////////////////////////////
//                MicroStar Version 1.01A           //
//                File Name : MS.PAS                //
////////////////////////////////////////////////////}
{$C-,I-,K-}
program MicroStar;
{
  Copyright (c) 1985,86 by Borland International, Inc.

    List of data files:
    EDITERR.EDG    - Text file containing error messages.
}

{$I VARS.ED     }     { Constant, type, and variable
                      declarations }

procedure userStatusline(Var TWindow:Byte;
                         Column,line:Integer;
                         Insertflag:Insflag;
                         WW,AI: boolean); forward;
procedure usererror(Var Msgno:Byte);forward;
procedure pulldownmenu;forward;
procedure ClearSubMenu(ScreenPosition,Width:integer);
forward;

procedure printnext;forward;
function ErrorCheck (ErrorNum:byte) : boolean; forward;

{$I USER.ED     }     {Editor kernel and primitive level
                      helper routines }
{$I SCREEN.ED   }     {Screen updating routines }
{$I FASTCMD.ED  }     { Misc. command processing routines
                      that are not }
                      { overlaid because they are commonly
                      used and must be fast }

function ErrorCheck {(ErrorNum:byte) : boolean};
```

```
{  If the ErrorNum <> 0 then  there has been some kind
of error.  The appropriate error message is displayed,
and the user must press escape before the program will
resume execution.
}
var
  ch    : char;
  st    : VarString;
  count : byte;
begin { function ErrorCheck }
  IOsave := ErrorNum;                { assign the error
                                     number to the    }
  if IOsave <> 0 then                { global variable
                                     IOsave }
  begin
    MenuColor := PLowColor;          { Update the text
                                     color }
    Beep;                            { Beep to get the
                                     user's attention }
    SetMemAddress(1,1);              { set the display
                                     coordinates to 1,1}
    st := EditMessage(IOSave);       { Get the appropriate
                                     error message }
    st := st + ' - Press ESC ';
    SaveScreen(SaveCmdLine,80);      { save the current
                                     command line text}
    WriteString(st);                 { display the error
                                     message }
     for count := (EndPosition - 2) downto byte(St[0])
         do { pad line w/ spaces}
      WriteChar(32);
    repeat                           { read the keyboard
                                     for ESCAPE }
      Read(kbd,ch)
    until (ch = #27) and not keypressed;
    SetMemAddress(1,1);              { set the display
                                     coordinates to 1,1}
```

```
    RestoreScreen(SaveCmdLine,80); { restore original
                                     text on command  }
  end;                               { line }
  ErrorCheck := (IOSave <> 0);
end;  { function ErrorCheck }

{$I PRINT.ED }  {Printing routines}

procedure UserReplace(Var Ch:byte);
{  First a window is created, then the string is
displayed in the window and the user is prompted for a
character
}
const
  prompt : string[13] = 'Replace (Y/N):';
begin { procedure UserReplace }
  ch := byte (BoolPrompt (10,prompt));
end;  { procedure UserReplace }

{********** The overlay area starts here **********}
{$I INIT.ED} { Overlaid version of initialization code }
{$I KCMD.ED} { Overlaid version of Ctrl-K routines }
{$I OCMD.ED} { Overlaid version of Ctrl-O routines }
{$I QCMD.ED} { Overlaid version of Ctrl-Q routines }
{$I CMD.ED } { Overlaid version of general editing
               commands }
{$I MSCMD.ED} { Overlaid MicroStar-specific commands }
(*  not used:
{$I SPELL.ED} { Microstar/Lightning spelling checker
                (overlaid) }
*)
{********** The overlay area ends here **********}

procedure SetOnOff(var c; state: boolean); forward;

procedure GetFlags;
{  Set the current text formating flags }
```

```
begin
  with Curwin^ do
  begin
    InsertDef := Not boolean (InsertFlag);
    SetOnOff(st41,InsertDef);
    AutoinDef := AI;
    SetOnOff(st43,AutoinDef);
    WordWpdef := WW;
    SetOnOff(st42,WordWpDef);
  end;
end;

procedure CloseWindow;
{  If the second  window is open, the information in it
is deleted and the boolean flag indicating the windows
status is set to false.  The F10 function key prompt is
then displayed in place of Window Two's status line.
Note that windows 2 and three are the two TEXT windows
available in MicroStar.  Window 1 is used for the status
line window and pull down menus.
}
begin { procedure CloseWindow }
  if not SecondWindow then     { can't do with one
                                 window }
    if ErrorCheck (121) then exit;
  CurrWindow := 2;             { update current window to
                                 window two }
  EditWindowDelete(3);         { delete the text in
                                 window three }
  SecondWindow := false;       { update boolean flag }
  DispF10;                     { display F10 key prompt }
  GetFlags;
  UpdCurFlag := true;
end;  { procedure CloseWindow }

procedure ToggleWindow;
```

```
{  If both windows are currently defined, the cursor is
moved from the current window to the alternate.
}
begin { procedure ToggleWindow }
  if not SecondWindow then  { can't do with one window }
    if ErrorCheck(121) then exit;
  if CurrWindow = 3 then    { toggle current window }
    Currwindow := 2
  else
    Currwindow := 3;
  EditWindowGoto(CurrWindow); { update current window's
                              cursor, etc. }
  GetFlags;
end;  { procedure ToggleWindow }

procedure OpenWindow;
{  First a window is created on the screen and a prompt
is displayed in it asking for the number of lines of the
screen to be used for the second window.
}
const
  st:String[25] = 'How many lines (3 - 17): ';
  cl = 40; ln = 7; wd = 34; hg = 3;
var
  xy : byte;
  ok : boolean;
begin { procedure OpenWindow }
  if SecondWindow then     { if 2 windows open, error }
    if ErrorCheck(120) then exit;
  MakeWindow(cl,ln,wd,hg); { make prompt window }
  ok := GetInteger(WindowLines,3,17,cl + 2,ln +
1,st,dummyst);
  RestoreWindow(cl,ln,wd,hg); { restore screen
                              contents }
  if ok then
  begin
```

```
      EditWindowCreate(WindowLines+1,2);   { allocate a
                                           text window }
      EditWindowLink(2,3);                 { link the new
                                           window w/ the
                                           text }
      SetMemAddress(1,25 - Windowlines);   { set display
                                           address on
                                           screen } /
      MenuColor := PFrameColor;            { set display
                                           color }
      WriteChar(LConnect);                 { display window
                                           frame }
      for xy := 2 to 79 do
        WriteChar(Horizontal);
      WriteChar(UpperRight);
      SecondWindow := true;                { set global
                                           flag to true }
      with Curwin^.Fwdlink^ do             { the new window
                                           is next on
                                           list }
      begin                                { set its flags
                                           like current
                                           ones }
        Insertflag := Insflag(not Insertdef);
        WW := WordWpdef;
        AI := AutoIndef;
      end;
      ToggleWindow;                        { go to it right
                                           away }
    end;
  end;  { procedure OpenWindow }

  procedure SetLoggedDrive;
  {  Sets a new logged drive/directory }
  const
    cl = 1; ln =11; wd = 80; hg = 4;
    prompt : String[20] = 'New Drive/Directory:';
```

```
var
  NewFilePath : fstr66;
begin { procedure SetLoggedDrive }
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFilter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(prompt);
  MenuColor := WNormColor;
  NewFilePath := GetString(cl + 23,ln +
1,54,FilePath,true);
  If NewFilePath = #27 then NewFilePath := '';
  if NewFilePath <> '' then
  begin
    ChDir(NewFilePath);              { try to change to
                                     new directory }
    if IOResult <> 0 then
    begin
      if ErrorCheck(122) then;
    end else
    begin
      GetDir(0,FilePath);            { if no errors, use
                                     DOS to get std form }
      if not ErrorCheck(IOResult) then;
    end;
  end;
  RestoreWindow(cl,ln,wd,hg);
end; { procedure SetLoggedDrive }

procedure ProcessFileName(var FileName : fstr66);
{  Takes the filename and adds the directory path, etc.
to it }
var
  i : integer;
  TmpDir : fstr66;
begin { procedure ProcessFileName }
  if FileName = '' then exit;
```

```
  if (length(FileName) = 1) or
     ((length(FileName) >= 2) and (FileName [2] <> ':'))
then
        FileName := Copy(FilePath,1,2) + FileName;
  if length(FileName) >= 3 then
    if FileName [3] <> '\' then
    begin
      GetDir(Succ(Ord(FileName[1]) - Ord('A')),TmpDir);
      if IOResult <> 0 then
      begin
        if ErrorCheck(122) then;
        FileName := '';
      end else
        begin
        if TmpDir[length(TmpDir)] <> '\' then
          TmpDir := TmpDir + '\';
        Insert(Copy(TmpDir,3,length(TmpDir) -
2),FileName,3);
        end;
    end;
end; { procedure ProcessFileName }

function WriteFile(BlockOnly: boolean): boolean;
{  Writes a block/file of text to disk }
const
  cl = 1; ln = 10; wd = 80; hg = 4;
  blockmsg: String[31] = 'Writing the block. Please wait
';
  oldfile: String[31] =  'File exists. Continue (Y/N)?
';
var
  i : byte;
  StPtr: ^fstr66;
  fl: file;
  ch: char;
  p1,p2: Plinedesc;
  SaveCursorMode: integer;
```

```
begin { procedure WriteFile }
  WriteFile := false;
  if BlockOnly then
    if BlockHide or (BlockFrom = nil) or (BlockTo = nil)
then
      if ErrorCheck (26) then exit;
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFilter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(fileprompt);
  MenuColor := WNormColor;
  if BlockOnly then
    StPtr := Addr(BlockFileName)
  else
    StPtr := Addr(WriteFileName);
  StPtr^ := GetString(cl + 12,ln + 1,66,StPtr^,true);
  If StPtr^ = #27 then StPtr^ := '';
  ProcessFileName(StPtr^);
  if StPtr^ <> '' then
  begin
    ch := 'Y';
    Assign(fl,StPtr^);
    reset(fl);
    if IOResult = 0 then
    begin
      MenuColor := WLowColor;
      SetMemAddress(cl + 2,ln + 2);
      WriteString(oldfile);
      repeat ch := Upcase(readChar); until Ch in
['Y','N', #27];
      close(fl);
    end;
    if ch = 'Y' then
    begin
```

```
        MenuColor := WLowColor;
        SetMemAddress(cl + 2, ln + 2);
        if BlockOnly then begin
          p1 := BlockFrom;
          p2 := BlockTo;
          WriteString(blockmsg);
        end
        else
        begin
          WriteString(filemsg);
          p1 := Curwin^.Topline;
          while p1^.Backlink <> nil do  { Search for
beginning of stream }
            p1 := p1^.Backlink;
          p2 := nil;       { Last line has nothing after }
        end;
        if EditBlockWrite(StPtr^, p1, p2) then
        begin
          WriteFile := true;
          if not BlockOnly then
          begin
            EditChangeFlag := false;
            WorkFileName := WriteFilename;
            NoWorkFile := false;
            dispFileName;
          end;
        end;
      end;
    end;
    RestoreWindow(cl,ln,wd,hg);
    SetCursorMode(SaveCursorMode);
  end; { procedure WriteFile }

  function OKtoAbandon: boolean;
  {  Displays the user prompt to abandon file changes or
not }
  const
```

```
  prompt : String[23] = 'Abandon changes (Y/N)? ';
var
  ch:char;
begin { function OKtoAbandon }
  ch := 'Y';
  if EditChangeFlag then
    ch := BoolPrompt(10,prompt);
  OKtoAbandon := (ch = 'Y');
end; { function OKtoAbandon }

procedure ClearFile;
{  Delete the text from the current window }
begin { procedure ClearFile }
  ClearSubmenu(SubMenuPos,Mem[Cseg:MenuData
    [CurrSubMenu].SubStart]);
  if SecondWindow then CloseWindow;
  EditWindowDeleteText;
  noworkfile := true;
  EditChangeFlag := false;
  UpdCurFlag := true;
  EditUpdPhyscr;
end; { procedure ClearFile }

function Abandonfile: boolean;
{  Abandon the file being edited }
begin
  AbandonFile := false;
  if OKtoAbandon then
  begin
    ClearFile;
    AbandonFile := true;
  end;
end; {AbandonFile}

function Savefile : boolean;
{  Save the current file to disk }
const
```

```
  cl = 15; ln = 10; wd = 34; hg = 3;
var
  i  : Byte;
  st : fstr66;
  p  : Plinedesc;
  SaveCursorMode: integer;
begin { procedure SaveFile }
  if NoWorkFile then
  begin
    SaveFile := WriteFile(false);
    exit;
  end;
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  SaveFile := false;
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(filemsg);
  if (Pos('.BAK',WorkFilename) = 0) and FileExist then
  begin
    i := Pos('.',WorkFilename);
    if i = 0 then st := WorkFilename + '.BAK'
             else st := Copy(WorkFilename,1,i) + 'BAK';
    i := CopyFile(false,WorkFilename,st);
    if i < 0 then if errorcheck(104) then;
  end;
  p := Curwin^.Topline;
  while p^.Backlink <> nil do  { Search for beginning of
stream }
    p := p^.Backlink;
  if EditBlockWrite(WorkFilename, p, nil) then
  begin
     EditChangeFlag := false;
     SaveFile := true;
  end;
  RestoreWindow(cl,ln,wd,hg);
```

```
  SetCursorMode(SaveCursorMode);
end;  { procedure SaveFile }

procedure ExitFile;
{  Exit the editor after checking if the user wants to
halt printing/save changes
}
const
  StopPrint : String[15] = 'Abort Printing (Y/N)?';
  SaveIt : String [19] = 'Save Changes (Y/N)?';
var
  ch: char;
begin { procedure ExitFile }
  ch := 'N';
  if EditChangeFlag then
    ch := boolprompt(10,SaveIt);
  if ch = #27 then exit;
  if ch = 'Y' then
   if not SaveFile then exit;
  if printing then
  begin
    ClearFile;
    if boolprompt(10,StopPrint) <> 'Y' then exit;
  end;
  printexit;
  EditExit;
  ClrScr;
  SetCursorMode(CursorOn);
  Halt;
end;  { procedure ExitFile }

procedure ReadBlock;
{  Read a block into the current window }
const
  cl = 1; ln = 10; wd = 80; hg = 4;
  proceed: String[29] = 'Reading the file. Please wait';
var
```

```
    i : byte;
    st: fstr66;
    fl: file;
    ch: char;
    SaveCursorMode: integer;
  begin { procedure ReadBlock }
    SaveCursorMode := GetCursorMode;
    SetCursorMode(Cursoroff);
    MakeWindow(cl,ln,wd,hg);
    MenuColor := WLowColor;
    CharFilter := FileCharFIlter;
    SetMemAddress(cl + 2, ln + 1);
    WriteString(fileprompt);
    MenuColor := WNormColor;
    BlockFilename := GetString(cl + 12,ln +
  1,66,BlockFilename,true);
    If BlockFilename = #27 then BlockFilename := '';
    ProcessFileName(BlockFileName);
    if BlockFilename <> '' then
    begin
      MenuColor := WLowColor;
      SetMemAddress(cl + 2, ln + 2);
      WriteString(proceed);
      EditFileRead(BlockFilename);
    end;
    RestoreWindow(cl,ln,wd,hg);
    SetCursorMode(SaveCursorMode);
  end;  { procedure ReadBlock }

  procedure OpenFile(prompting: boolean);
  {  Opens a new file specified by the user }
  const
    cl = 1; ln = 9; wd = 80; hg = 4;
    prompt : String[10] = 'Open File:';
    proceed: String[29] = 'Reading the file. Please wait';
    newfile: String[29] = 'File is new. Continue (Y/N): ';
  var
```

```
  fl: file;
  ch: char;
  SaveCursorMode: integer;
begin { procedure OpenFile }
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFilter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(prompt);
  MenuColor := WNormColor;
  if prompting then
    WorkFilename := GetString(cl + 13,ln +
1,66,WorkFilename,true)
  else begin
    SetMemAddress(cl + 13, ln + 1);
    WriteString(WorkFileName);
  end;
  If WorkFilename = #27 then WorkFilename := '';
  ProcessFileName(WorkFileName);
  if WorkFilename <> '' then
  begin
    ch := 'Y';
    Assign(fl,WorkFilename);
    if not ErrorCheck(IOResult) then
    begin
      reset(fl);
      if IOResult = 0 then
      begin
        close(fl);
        FileExist := true;
        MenuColor := WLowColor;
        SetMemAddress(cl + 2, ln + 2);
        WriteString(proceed);
        EditFileRead(WorkFilename);
        FileExist := true;
```

```
          NoWorkfile := false;
        end
        else
        begin
          MenuColor := WLowColor;
          SetMemAddress(cl + 2,ln + 2);
          WriteString(newfile);
          repeat ch := Upcase(readChar); until Ch in
['Y','N'];
          if ch = 'Y' then
          begin
            NoWorkfile := false;
            FileExist := false;
          end;
        end;
      end;
    end;
    dispFileName;
    EditchangeFlag := false;
    RestoreWindow(cl,ln,wd,hg);
    UpdCurflag := true;
    SetCursorMode(SaveCursorMode);
  end;  { procedure OpenFile }

  procedure CloseFile;
  {  Closes the current file after asking the user if the
  changes should be saved.
  }
  const
    prompt : String[20] = 'Save changes (Y/N)? ';
  var
    ch:char;
  begin { procedure CloseFile }
    if EditChangeFlag then
    begin
      ch := BoolPrompt(10, prompt);
      if ch = #27 then exit;
```

```
    if ch = 'Y' then
      if not SaveFile then exit;
  end;
  ClearFile;
  OpenFile(true);
end;  { procedure CloseFile }

Procedure ToggleInsert;
{  This routine toggles the flags indicating if the user
is in AutoInsert mode.  It also updates the menu and
screen.
}
begin { procedure ToggleInsert }
  EditToggleInsert;         { toggle window's flag }
  GetFlags;                 { update global flags  }
  EditUpdwinsl(CurWin);     { fix status line, too }
end;  { procedure ToggleInsert }

Procedure ToggleAutoIndent;
{  This routine toggles the flags indicating if the user
is in AutoIndent mode.  It also updates the menu.
}
begin { procedure ToggleAutoIndent }
  EditToggleAutoindent;       { toggle window's flag }
  GetFlags;                   { update global flags }
end;  { procedure ToggleAutoIndent }

Procedure ToggleWordwrap;
{  This routine toggles the flags indicating that the
user is in WordWrap mode.  It also updates the menu and
screen.
}
begin { procedure ToggleWordWrap }
  EditToggleWordwrap;      { toggle window's flag }
  GetFlags;                { update global flags  }
  EditUpdwinsl(CurWin);    { fix status line, too }
end;  { procedure ToggleWordWrap }
```

```
Procedure LeftMargin;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[17] = 'New left margin: ';
  cl = 34;                    { window coordinates }
  ln = 9;
  wd = 26;
  hg = 3;
begin { procedure LeftMargin }
  MakeWindow(cl,ln,wd,hg);    { make the prompt window }
  if GetInteger(LMargindef,0,Rmargindef-1,cl + 2,ln +
1,st,st44) then
    EditSetLeftMargin(LMargindef); { set new left
                                   margin }
  RestoreWindow(cl,ln,wd,hg);      { restore original
                                   screen }
end;  { procedure LeftMargin }

procedure RightMargin;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[18] = 'New right margin: ';
  cl = 34;                    { window coordinates }
  ln = 10;
  wd = 27;
  hg = 3;
begin { procedure Rightmargin }
  MakeWindow(cl,ln,wd,hg);              { create the
                                        prompt window }
  if GetInteger(RMargindef,LMargindef+1,MaxLinelength,cl
+ 2,ln + 1,st,st45) then
```

```
    EditSetRightMargin(RMargindef);      { set the new
                                         right margin }
  RestoreWindow(cl,ln,wd,hg);          { restore original
                                       screen  }
end;  { procedure SetRightMargin }

procedure DefineTab;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[14] = 'New tab size: ';
  cl = 34;                           { window coordinates }
  ln = 11;
  wd = 23;
  hg = 3;
begin { procedure DefineTab }
  MakeWindow(cl,ln,wd,hg);            { display the
                                      window }
  if GetInteger(TabDef,1,255,cl + 2,ln + 1,st,st46) then
    EditDefineTab(Tabdef);            { define new tab
                                      width }
  RestoreWindow(cl,ln,wd,hg);         { restore original
                                      screen }
end;  { procedure DefineTab }

procedure SetUndoLimit;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[16] = 'New undo limit: ';
  cl = 34;                           { window coordinates }
  ln = 12;
  wd = 25;
  hg = 3;
begin { procedure SetUndoLimit }
```

```
  MakeWindow(cl,ln,wd,hg);          { display the window }
  if GetInteger(Undodef,1,255,cl + 2,ln + 1,st,st47)
then
    EditSetUndolimit(Undodef);     { sets new undo limit}
  RestoreWindow(cl,ln,wd,hg);      { restore old screen }
end;  { procedure SetUndoLimit }

procedure GotoLine;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[12] = 'Which line: ';
  cl = 27;                         { window coordinates }
  ln = 8;
  wd = 21;
  hg = 3;
begin { procedure GotoLine }
  MakeWindow(cl,ln,wd,hg);         { display window      }
  if GetInteger(Targetline,1,30000,cl + 2,ln +
1,st,dummyst) then
    EditGotoline(Targetline);      { goto the line       }
  RestoreWindow(cl,ln,wd,hg);      { restore screen      }
end;  { procedure GotoLine }

procedure GotoColumn;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[14] = 'Which column: ';
  cl = 27;                          { window coordinates }
  ln = 9;
  wd = 23;
  hg = 3;
begin { procedure GotoColumn }
  MakeWindow(cl,ln,wd,hg);          { display window  }
```

```
  if GetInteger(TargetColumn,1,255,cl + 2,ln +
1,st,dummyst) then
    EditGotoColumn(TargetColumn); { goto the column }
  RestoreWindow(cl,ln,wd,hg);     { restore screen  }
end;  { procedure GotoColumn }

procedure GotoMarker;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[14] = 'Which marker: ';
  cl = 27;                        { window coordinates }
  ln = 10;
  wd = 23;
  hg = 3;
begin { procedure GotoMarker }
  MakeWindow(cl,ln,wd,hg);        { display window }
  if GetInteger(TargetMarker,1,Maxmarker,cl + 2,ln +
1,st,dummyst) then
    EditJumpMarker(TargetMarker);{ jump to marker }
  RestoreWindow(cl,ln,wd,hg);     { restore screen }
end;  { procedure GotoMarker }

procedure SetMarker;
{  This routine sets a text marker at the current cursor
position }
const
  st:String[15] = 'Select marker: ';
  cl = 27;                        { window coordinates }
  ln = 11;
  wd = 24;
  hg = 3;
begin { procedure SetMarker }
  MakeWindow(cl,ln,wd,hg);        { display window     }
  if GetInteger(TargetMarker,1,Maxmarker,cl + 2,ln +
1,st,dummyst) then
```

```
    EditSetMarker(TargetMarker); { define text marker }
  RestoreWindow(c1,ln,wd,hg);    { restore screen     }
end;  { procedure SetMarker }

{$I PULLDOWN.ED}              { Pulldown menu routines }

function SecondChar(st:st40):Byte;
var
  ch:Byte;
begin { function SecondChar }
  MenuColor := PLowColor;
  SetmemAddress(1,1);
  SaveScreen(SaveCmdLine,80);
  WriteString(St);
  MenuColor := PNormColor;
  for ch := EndPosition - 2 downto Byte(St[0]) do
WriteChar(32);
  SecondChar := EditCtrlchar;
  SetmemAddress(1,1);
  RestoreScreen(SaveCmdLine,80);
end;  { function SecondChar }

{ INCLUDE MICROSTAR PREFIXED COMMAND DISPATCHERS }
{$I K.ED}
{$I O.ED}
{$I Q.ED}

procedure UserStatusline{(Var TWindow:Byte;
Column,line:Integer;
                  Insertflag:Insflag; WW,AI: boolean)};
{ This routine updates the UserStatus line for the
current window }
const
  Statusline :  String[21] = 'Line:    1 Col:    1 ';
  is : String[2] = 'IN';
  nn : String[2] = '  ';
var
```

```
  st:string[5];
begin { procedure UserStatusLine }
  if TWindow <> 1 then
  begin
    SetMemAddress(58,TWindow);
    If TWindow = CurrWindow then
      MenuColor := WLowColor
    else
      MenuColor := WNormColor;
    Str(line:5,st);
    move(St[1],Statusline[6],5);
    Str(Column:5,st);
    move(St[1],Statusline[16],5);
    WriteString(StatusLine);
    if TWindow = CurrWindow then   { show flags for
                                   current only }
    begin
      MenuColor := WLowColor;      { set the display
                                   color }
      SetMemAddress(53,2);         { set address of
                                   status area }
      if boolean (Insertflag) then { enum type => false
                                   if insert }
        WriteString(nn)            { if no insert,
                                   display blanks }
      else
        WriteString(is);           { if insert then
                                   display IN }
      SetMemAddress(53,3);
      if WW then
        WriteString(Stww)          { if true then
                                   display WW }
      else
        WriteString(nn);           { else display
                                   blanks }
    end;
  end;
```

```
  TWindow := 0;
end;  { procedure UserStatusLine }

procedure Init;
{  The copyright is displayed, the workfile name is
requested, and all global variables, data structures,
etc. are initialized.
}
const st1:string[44] = ' EDIT, System Editor';
      st2:String[44] = ' 3D Systems SLA-1
Stereolithography System';
      st3:String[44] = ' Version 3.00   3D Systems,
                                        Inc.';
var fl:file;
     i:Byte;
    ParmUsed : boolean;

  procedure DisplayCopyRight;
  begin { procedure DisplayCopyRight }
    MakeWindow(1,1,47,6);
    MenuColor := WNormColor;
    SetmemAddress(3,2);
    WriteString(st1);
    SetmemAddress(3,3);
    WriteString(st2);
    SetmemAddress(3,5);
    WriteString(st3);
  end;  { procedure DisplayCopyRight }

  procedure DisplayLineStatWindow;
  begin { procedure DisplayLineStatWindow }
    MakeWindow(51,1,30,4);
    SetMemAddress(56,1);
    WriteChar(TConnect);
    SetMemAddress(56,2);
    WriteChar(Vertical);
    SetMemAddress(56,3);
```

```
    WriteChar(Vertical);
    SetMemAddress(56,4);
    WriteChar(LRConnect);
  end;  { procedure DisplayLineStatWindow }

procedure OverlayError;
{  This routine causes the program to abort because an
error was encountered when trying to locate the overlay
file.
}
begin
  Writeln('Overlay file not found.  Program aborted.');
  halt;
end; { OverlayError }

begin { procedure Init }
  assign(f1,'\3DSYS\EDITOR\EDIT.000');  { check for
                                        overlay file }
  if IOresult <> 0 then OverlayError;
  reset(f1);
  if IOresult <> 0 then OverlayError;
  close(f1);
  if IOresult <> 0 then OverlayError;
  FilePath := '\3DSYS\EDITOR';
  if ErrorCheck(IOresult) then halt;
  OvrPath(FilePath);
  if ErrorCheck(IOresult) then halt;
  ErrorFile := DefErrorFile;
  if FilePath[length(FilePath)] <> '\' then  { same for
                                             error
                                             message
                                             file }
    ErrorFile := '\' + ErrorFile;
  ErrorFile := FilePath + ErrorFile;
  GetDir(0,FilePath);              { always look in
                                   right dir }
```

```
CursorOn := GetCursorMode;          { assume the cursor
                                    was on when we
                                    entered    }
EditInitialize;                     { call EDITOR TOOLBOX
                                    initialization
                                    routine }

EditWindowCreate(22,1);
EditWindowGoto(2);
DisplayCopyRight;
SecondWindow := false;
XPosition := 1;
YPosition := 1;
CurrWindow := 2;
ParmUsed := ParamCount <> 0;
if ParmUsed then  { check if filename was entered at
                  command line }
begin
  WorkFileName := ParamStr(1);
  for i := 1 to Byte(WorkFilename[0]) do
    WorkFilename[i] := Upcase(WorkFilename[i]);
end;
EditChangeFlag := false;
NoWorkFile := true;
gotoxy(1,5);       {Give cursor a reasonable initial
                   position}
OpenFile(not ParmUsed);
for i := 5 to 6 do
begin
  SetMemAddress(1,i);
  RestoreScreen(WArray[pred(i),0],47);
end;
DisplayLineStatWindow;
InitMainmenu;
ExitPulldown;
if boolean (Curwin^.Insertflag) = Insertdef then
ToggleInsert;
if Curwin^.WW <> WordWpdef  then ToggleWordwrap;
```

```
  if Curwin^.AI <> AutoinDef then ToggleAutoindent;
  DispF10;
  EditSetLeftMargin(LMargindef);
  EditSetRightMargin(RMargindef);
  EditDefineTab(Tabdef);
  EditSetUndolimit(Undodef);
end;  { procedure Init }

procedure UserCommand(var ch : byte);
{  This is the user command processor hook }
begin { procedure UserCommand }
  if ch = 0 then {Function key}
     begin
     ch := EditGetInput;
     if ch = 68 then {F10 pulls down menu}
        PulldownMenu;
     ch := 255; {Always absorb keycode}
     end;
end;  { procedure UserCommand }

procedure UserError {(var Msgno : byte)};
{  This is the user error handler hook }
begin { procedure UserError }
  If ErrorCheck(Msgno) then;
  Msgno := 0;
end;  { procedure UserError }

procedure UserTask;
{  This is the user BackGround task hook }
begin { procedure UserTask }
  Printnext;
end;  { procedure UserTask }

{$I DISP.ED   }     { MicroStar main command
                    dispatcher }
{$I TASK.ED   }     {scheduling subsystem and central
                    dispatcher }
```

```
{$I INPUT.ED  }      {input routines }

begin  { main program block }
  Init;
  EditSystem;  { Use the default main loop }
  ClrScr
end.
^Z
```

```
{

   LibNC
   Library of No-Math Coprocessor Routines
   3D Systems Stereolithography System

   for use in all versions of STEREO and BUILD
      * STEREOC       * BUILDC
      * STEREONC      * BUILDNC

}

unit LibNC;

{$R-,S-,I-,D-,T-,F-,V-,B-,L+ }
{$M 16384,0,655360 }

{$N-}
interface

function LibRound(num : Real) : LongInt;
function LibTrunc(num : Real) : LongInt;

implementation

function LibRound(num : Real) : LongInt;
begin
  LibRound := Round(num);
end;

function LibTrunc(num : Real) : LongInt;
begin
  LibTrunc := Trunc(num);
end;

end.^Z
```

```
{=========================================================

    The LASER Unit
    LASER.PAS Source, LASER.TPU Unit Object

    StereoLithography Laser Controller Software
Interrupt Interface Routines

    for 3D Systems, Inc.

    Version 3.20  by Tarnz Technologies

=========================================================
```

The purpose of these routines is to interface general-purpose Turbo Pascal programs with the memory resident laser controller software. The machine language-based laser controller software the routines below interface to can be used with other languages like Basic, C, and 8086 Assembly Language.

Below is a list of the routines in this unit. Most of the functions relate directly to specific software interrupt calls. The Code is the related interrupt's AX (AH,AL) value. Some of the interrupt functions require arguments to be passed in other registers, and some will return results. See the actual routines for more details on the interface protocol.

Pascal Function	Code	Description

__LaserOk		see if laser software is installed
__LaserPresent		tell if laser software not installed
__OpenLaser	00 --	open laser DAC channel and initialize
__CloseLaser	01 --	close laser DAC channel and restore system
	02 xx	add command to background command queue
__AddVector	02 00	add vector command
__AddLongVector	02 01	add long vector command (large deltas)
__AddOpenShutter	02 02	add open shutter command
__AddCloseShutter	02 03	add close shutter command
__AddChangeTimebase	02 04	add change time base command
__AddPause	02 05	add pause command
__ResetCmdQueue	03 --	reset background command queue
__GetCmdQueueSize	04 --	get background command queue size
	05 xx	get option
__GetIntNum	05 00	get software interrupt vector number
__GetBkgndTime	05 01	get background task interval time
__GetControlWord	05 02	get laser control word
__AutoShutterModeOn	05 02	get status of auto shutter mode
__PosAckModeOn	05 02	get status of position acknowledge mode
__BlockModeOn	05 02	get status of block mode
__GetStatus	05 03	get laser status

__GetOpenStatus	05 03	get laser channel open status
__GetBkgndIdleStatus	05 03	get background idle status
__GetTurnOffDelay	05 04	get laser turn-off delay
__GetTurnOnDelay	05 05	get laser turn-on delay
__GetHomePosition	05 06	get laser home position
	06 xx	set option
__SetIntNum	06 00	set software interrupt vector number
__SetBkgndTime	06 01	set background task interval time
__SetControlWord	06 02	set laser control word (general)
__SetAutoShutMode	06 02	set laser control word, bit 0
__SetPosAckMode	06 02	set laser control word, bit 1
__SetBlockMode	06 02	set laser control word, bit 2
	06 03	-
__SetTurnOffDelay	06 04	set laser turn-off delay
__SetTurnOnDelay	06 05	set laser turn-on delay
__SetHomePosition	06 06	set laser home position
	07 xx	background task execution control
__Execute	07 00	execute command queue to current point and stop
__ExecuteClear	07 01	execute command queue to current point, stop, and clear
__Clear	07 02	clear command queue (reset queue's start pointer)
	08 xx	background task redraw control

```
__SetStepPeriods      08 00    set multi-pass step
                               periods
__SetRedrawCount      08 01    set redraw pass count
__SetRedrawDelay      08 02    set redraw delay
__Mark                08 03    mark redraw start position
__Redraw              08 04    perform multi-pass redraw

                      09 xx    multiple timers control
__GetTimerCapability  09 00    get multiple timer
                               capability
__SelectTimerChannel  09 01    select timer channel
__GetChannelTimebase  09 02    get timer channel timebase
                               (real-time)
__GetCurrentTimebase  09 03    get current timer timebase
                               (real-time)

================================================== }

{$R-,S-,I-,D-,T-,F-,V-,B-,N-,L- }
{$M 16384,0,0 }

unit Laser;

interface

uses Dos;

const
  _LASERVECTOR        =   $64;       { Interrupt vector }
  _OK                 =     0;       { Errors returned by
                                       Level Zero }
  _INTERNAL_ERROR     =   255;
  _MINIMUM_TICKS      =    59;       { don't go below 50
                                       us }
  _LASER_QUEUE_MAX    = 60000.0;     { Laser Queue size }
```

```
  __ALLBITS = $FFFF;
  __BITVALUE: array[0..15] of Word =
    ($0001, $0002, $0004, $0008, $0010, $0020, $0040,
$0080, $0100, $0200, $0400, $0800, $1000, $2000, $4000,
$8000);

type
  _Str16  = string[16];
  _VectorType = record
      fractX: Byte;      { start coordinates }
      startX: Word;
      fractY: Byte;      { fractional portion - 1 byte,
                         should be set to 0 }
      startY: Word;      { integer portion    - 2
                         bytes }
      deltaX: Integer;   { +/- change values in 1/256
                         units }
      deltaY: Integer;
        incr: Word;      { # of increments within vector
                         count }
      end;
  _LongVectorType = record
      fractX: Word;      { start coordinates }
      startX: Word;
      fractY: Word;      { fractional portion - 1 byte,
                         should be set to 0 }
      startY: Word;      { integer portion    - 2
                         bytes }
      deltaX: LongInt;   { +/- change values in 1/65536
                         units }
      deltaY: LongInt;
        incr: Word;      { # of increments within vector
                         count }
      end;
  _StepPeriodsType = record
      num    : Byte;
      periods: array[0..9] of Word;
```

```
    end;

var
  __IntReg: Registers;
  LaserVersion: _Str16;

function __LaserOk: Boolean;
function __LaserPresent: Boolean;
function __OpenLaser(ticks: Word): Integer;
function __CloseLaser: Integer;
function __AddVector(startX,startY: Word;deltaX,deltaY:
Integer;
                     incr: Word): Integer;
function __AddLongVector(startX,startY:
Word;deltaX,deltaY: LongInt;
                     incr: Word): Integer;
function  __AddOpenShutter: Integer;
function __AddCloseShutter: Integer;
function __AddChangeTimebase(NewTicks: Word): Integer;
function __AddPause(DelayTicks: Word): Integer;
function __ResetCmdQueue: Integer;
function __GetCmdQueueSize(var size: Word): Integer;
function __GetIntNum(var num: Word): Integer;
function __GetBkgndTime(var ticks: Word): Integer;
function __GetControlWord(var ctrl: Word): Integer;
function __AutoShutterModeOn: Boolean;
function __PosAckModeOn: Boolean;
function __BlockModeOn: Boolean;
function __GetStatus(var status1,status2: Word):
Integer;
function __GetOpenStatus: Boolean;
function __GetBkgndIdleStatus: Boolean;
function __GetTurnOffDelay(var ticks: Word): Integer;
function __GetTurnOnDelay(var ticks: Word): Integer;
function __GetHomePosition(var X,Y: Word): Integer;
function __SetIntNum(num: Word): Integer;
function __SetBkgndTime(ticks: Word): Integer;
```

```
function __SetControlWord(ctrl: Word): Integer;
function __AlterControlWordBit(op: Char;bit: Word):
Integer;
function __SetAutoShutterMode(op: Char): Integer;
function __SetPosAckMode(op: Char): Integer;
function __SetBlockMode(op: Char): Integer;
function __SetTurnOffDelay(ticks: Word): Integer;
function __SetTurnOnDelay(ticks: Word): Integer;
function __SetHomePosition(X,Y: Word): Integer;
function __Execute: Integer;
function __ExecuteClear: Integer;
function __Clear: Integer;
function __SetStepPeriods(var sp_array:
_StepPeriodsType): Integer;
function __SetRedrawCount(count: Word): Integer;
function __SetRedrawDelay(delay: Word): Integer;
function __Mark: Integer;
function __Redraw: Integer;
function __GetTimerCapability: Boolean;
function __SelectTimerChannel(Channel: Byte): Integer;
function __GetChannelTimebase(Channel: Byte;var Ticks:
Word): Integer;
function __GetCurrentTimebase(var Ticks: Word): Integer;


implementation

{-------------------------------------------------------


    __LaserOk function
      check to see if laser controller code has been
installed.

    returns:
    false if code not installed
```

```
    true  if code is installed
    LaserVersion contains laser controller version
number

  ------------------------------------------------------
}

function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with __IntReg do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,__IntReg);
      CopyPtr := Ptr(Es,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __LaserOk := true;
      with __IntReg do
        CopyPtr := Ptr(Es,BX-46);
      move(CopyPtr^[1],LaserVersion[1],18);
      LaserVersion[0] := #4;
    end else begin
      __LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

{------------------------------------------------------

    __LaserPresent function
```

```
      same as LaserOk but also prints a message if laser
system is not installed.

    returns:
    false if code not installed
    true  if code is installed

  ------------------------------------------------- }


function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;


{-------------------------------------------------

    __OpenLaser(Ticks) function
      open up laser DAC channel with the background
server interrupting every Ticks timer 0 ticks.  Input
for timer 0 is a 1.19 MHz clock signal, so if Ticks =
119, the background task would activate once per 100 us;
with Ticks = 1190, once per ms; etc.

    returns error code

  -------------------------------------------------
}

function __OpenLaser(ticks: Word): Integer;
  begin
{ Min Time CHeck bypassed -- BE CAREFUL!!!!! }
```

```
{    if ticks<119 then ticks := 119; }
    with __IntReg do begin
      AX := $0000;
      BX := ticks;    (* LibTrunc(Period*1.19 {MHz}); *)
      Intr(_LASERVECTOR,__IntReg);
      __OpenLaser := AL;
    end;
  end;

{-------------------------------------------------------

    __CloseLaser function
      close off laser DAC channel after the background
command queue empties.  If laser needs to be stopped
immediately, __ResetVectorPool should be called before
this routine.

    returns error code

  ------------------------------------------------------- }


function __CloseLaser: Integer;
  begin
    with __IntReg do begin
      AX := $0100;
      Intr(_LASERVECTOR,__IntReg);
      __CloseLaser := AL;
    end;
  end;

{-------------------------------------------------------

    __AddVector(startX,startY,deltaX,deltaY,incr)
function add the vector to the background command queue.
Vector start coordinates are (startX,startY) and their
change value is (deltaX,deltaY).  The number of
```

```
incremental steps in the vector is incr.  deltaX and
deltaY are in a special format (high byte is +/- integer
portion; low byte is 1/256 units) conducive to high-
speed DAC updating.  See the 3DPC protocol specification
for more information.

    returns error code

  ------------------------------------------------
}

function __AddVector(startX,startY: Word;deltaX,deltaY:
Integer;
                     incr: Word): Integer;
  var
    VectorPacket: _VectorType;
  begin
    if incr>0 then begin
      VectorPacket.fractX := 0; { set up vector packet }
      VectorPacket.startX := startX;
      VectorPacket.fractY := 0;
      VectorPacket.startY := startY;
      VectorPacket.deltaX := deltaX;
      VectorPacket.deltaY := deltaY;
      VectorPacket.incr   := incr;
      with __IntReg do begin
        AX := $0200;
        ES := Seg(VectorPacket);    { point ES:BX to
                                    vector packet }
        BX := Ofs(VectorPacket);
        Intr(_LASERVECTOR,__IntReg);
        __AddVector := AL;
      end;
    end else __AddVector := 0;
  end;

{------------------------------------------------
```

```
    __AddLongVector(startX,startY,deltaX,deltaY,incr)
function add a long vector to the background command
queue.  Vector start coordinates are (startX,startY) and
their change value is (deltaX,deltaY).  The number of
incremental steps in the vector is incr.  deltaX and
deltaY are in a special format (high word is +/- integer
portion; low word is 1/65536 units).

    returns error code

---------------------------------------------------- }

function __AddLongVector(startX,startY:
Word;deltaX,deltaY: LongInt;
                     incr: Word): Integer;
  var
    VectorPacket: _LongVectorType;
  begin
    if incr>0 then begin
      VectorPacket.fractX := 0;     { set up vector
                                    packet }
      VectorPacket.startX := startX;
      VectorPacket.fractY := 0;
      VectorPacket.startY := startY;
      VectorPacket.deltaX := deltaX;
      VectorPacket.deltaY := deltaY;
      VectorPacket.incr   := incr;
      with __IntReg do begin
        AX := $0201;
        ES := Seg(VectorPacket);    { point ES:BX to
                                    vector packet }
        BX := Ofs(VectorPacket);
        Intr(_LASERVECTOR,__IntReg);
        __AddLongVector := AL;
      end;
    end else __AddLongVector := 0;
```

```
  end;

{-------------------------------------------------

    __AddOpenShutter function
      add the open shutter command to the background
command queue.  The shutter will be opened by the
background task when it gets to this command in the
command queue.

    returns error code

------------------------------------------------- }

function  __AddOpenShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0202;
      Intr(_LASERVECTOR,__IntReg);
      __AddOpenShutter := AL;
    end;
  end;

{-------------------------------------------------

    __AddCloseShutter function
      add the close shutter command to the background
command queue.  The shutter will be closed by the
background task when it gets to this command in the
command queue.

    returns error code

-------------------------------------------------}

function __AddCloseShutter: Integer;
  begin
```

```
    with __IntReg do begin
      AX := $0203;
      Intr(_LASERVECTOR,__IntReg);
      __AddCloseShutter := AL;
    end;
  end;

{-------------------------------------------------------

    __AddChangeTimebase(NewTicks) function
      add the change time base command to the background
command queue.  The timer that awakens the badckground
task will be reloaded with the count of NewTicks by the
background task when it gets to this command in the
command queue.

    returns error code

------------------------------------------------------- }

function __AddChangeTimebase(NewTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0204;
      BX := NewTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddChangeTimebase := AL;
    end;
  end;

{-------------------------------------------------------

    __AddPause(DelayTicks) function
      add the pause command to the background command
queue.  When the background task gets to this command,
it will pause for DelayTicks number of timer ticks.  If
the Background Interrupt Time is 119 (100 us), then a
```

```
pause of 1 ms will ensue if DelayTicks is 10.
Afterwards, background activity resumes.

    returns error code

-------------------------------------------------------}

function __AddPause(DelayTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0205;
      BX := DelayTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddPause := AL;
    end;
  end;
{-------------------------------------------------------

    __ResetCmdQueue function
      clear out the background command queue.  Usually
used as part of an abort procedure.  Best when used just
before a __CloseLaser call.

    returns error code

-------------------------------------------------------}

function __ResetCmdQueue: Integer;
  begin
    with __IntReg do begin
      AX := $0300;
      Intr(_LASERVECTOR,__IntReg);
      __ResetCmdQueue := AL;
    end;
  end;

{-------------------------------------------------------
```

```
    __GetCmdQueueSize function
      get the background command queue's size in actual
number of bytes.  Number of vectors is variable but can
be estimated by the number returned by this routine /
13.

    returns error code

 ------------------------------------------------------}

function __GetCmdQueueSize(var size: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0400;
      Intr(_LASERVECTOR,__IntReg);
      size := BX;
      if size<0 then size := size + 65536;
      __GetCmdQueueSize := AL;
    end;
  end;

{------------------------------------------------------

    __GetIntNum(num) function
      place in num the software interrupt number.  Not
really important unless we have a software interrupt
conflict.

    returns error code

 ------------------------------------------------------}

function __GetIntNum(var num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0500;
```

```
      Intr(_LASERVECTOR,__IntReg);
      num := BL;
      __GetIntNum := AL;
    end;
  end;

{-------------------------------------------------

    __GetBkgndTime(ticks) function
      place in time the background task interrupt
interval time in actual 1.19 MHz tick counts.  (119 is
100 us, 1190 is 1 ms, etc.)

    returns error code

 -------------------------------------------------}

function __GetBkgndTime(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0501;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetBkgndTime := AL;
    end;
  end;

{-------------------------------------------------

    __GetControlWord(ctrl) function
      place in ctrl the Laser Control Word showing the
status of various hardware and software mechanisms.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
```

```
            2       Block Mode
            3       -
            0       shutter control, 0=open/1=closed

    returns error code

 ------------------------------------------------------}

function __GetControlWord(var ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0502;
      Intr(_LASERVECTOR,__IntReg);
      ctrl := BX;
      __GetControlWord := AL;
    end;
  end;

{-------------------------------------------------------

    __AutoShutterModeOn function
      returns the status of the Auto Shutter Mode (true
for on).

 ------------------------------------------------------}

function __AutoShutterModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __AutoShutterModeOn := (ctrl and __BITVALUE[0]) > 0;

  end;

{-------------------------------------------------------
```

```
    __PosAckModeOn function
      returns the status of the Position Acknowledge
Mode (true for on).

-------------------------------------------------------}

function __PosAckModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __PosAckModeOn := (ctrl and __BITVALUE[1]) > 0;
  end;

{-------------------------------------------------------

    __BlockModeOn function
      returns the status of the Block Mode (true for
on).

-------------------------------------------------------}

function __BlockModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __BlockModeOn := (ctrl and __BITVALUE[2]) > 0;
  end;

{-------------------------------------------------------

    __GetStatus(status1,status2) function
```

```
      place in status1 and status2 the Laser system's
stati of various non-programmable hardware and software
mechanisms.

      word bit      description
      --------
-------------------------------------------------------
      status1.0     laser system channel open status
      status1.1     laser system software interrupt
                    established (sic)
      status1.2     laser system background task in idle
                    state
      status1.3-15  -
      status2.0     bit 0 of Port C1, X and Y-axis
                    position acknowledge
      status2.1     bit 1 of Port C1, Z-axis position
                    acknowledge
      status2.2     bit 2 of Port C1
      status2.3     bit 3 of Port C1
      status2.4-15  -

   returns error code

 -------------------------------------------------------}

function __GetStatus(var status1,status2: Word):
Integer;
  begin
    with __IntReg do begin
      AX := $0503;
      Intr(_LASERVECTOR,__IntReg);
      status1 := BX;
      status2 := CX;
      __GetStatus := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __GetOpenStatus function
      tells whether or not the laser system
communications channel has been opened and the timer
tick trap installed.  The function returns a boolean
value of TRUE if the channel is open.

    returns boolean value

 -------------------------------------------------------}

function __GetOpenStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetOpenStatus := (stat1 and
__BITVALUE[0]) > 0
      else __GetOpenStatus := false;
  end;

{-------------------------------------------------------

    __GetBkgndIdleStatus function
      tells whether or not the laser system background
task is currently in the idle state where no more laser
activity can be performed without some more
foreground-sourced commands.  The function returns a
boolean value of TRUE if the background task is idle.

    returns boolean value

 -------------------------------------------------------}

function __GetBkgndIdleStatus: Boolean;
```

```
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetBkgndIdleStatus := (stat1 and
__BITVALUE[2]) > 0
      else __GetBkgndIdleStatus := false;
  end;

{-----------------------------------------------------

    __GetTurnOffDelay(ticks) function
      get the idle state's laser turn-off delay in
interrupt tick counts.

    returns error code

-----------------------------------------------------}

function __GetTurnOffDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0504;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOffDelay := AL;
    end;
  end;

{-----------------------------------------------------

    __GetTurnOnDelay(ticks) function
      get the idle state's laser turn-on delay in
interrupt tick counts.

    returns error code
```

```
-------------------------------------------------------}

function __GetTurnOnDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0505;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOnDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __GetHomePosition(X,Y) function
      get the idle state's laser home position.

    returns error code

-------------------------------------------------------}

function __GetHomePosition(var X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0506;
      Intr(_LASERVECTOR,__IntReg);
      X := BX;
      Y := CX;
      __GetHomePosition := AL;
    end;
  end;


{-------------------------------------------------------

    __SetIntNum(num) function
```

```
      set the software interrupt number to num.  All
software interrupt call after this point must use this
new interrupt number.  The interrupt vector for the old
number is restored.

    returns error code

 ------------------------------------------------------}

function __SetIntNum(num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0600;
      BL := num;
      Intr(_LASERVECTOR,__IntReg);
      __SetIntNum := AL;
    end;
  end;

{------------------------------------------------------

    __SetBkgndTime(ticks) function
      set the background task interrupt interval to
ticks which has the actual number of 1.19 MHz tick
counts.  (119 is 100 us, etc.)

  returns error code}
------------------------------------------------------ }

function __SetBkgndTime(ticks: Word): Integer;
  begin
    if ticks<_MINIMUM_TICKS then ticks :=
_MINIMUM_TICKS;
    with __IntReg do begin
      AX := $0601;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
```

```
      __SetBkgndTime := AL;
    end;
  end;

{-------------------------------------------------

    __SetControlWord(ctrl) function
      set the Laser Control Word to ctrl.  It is
recommended that most software programs use the
__AlterControlWordBit and the specific mode set routines
which follow.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          4     shutter control, 0=open/1=closed

    returns error code

 -------------------------------------------------}

function __SetControlWord(ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0602;
      BX := ctrl;
      Intr(_LASERVECTOR,__IntReg);
      __SetControlWord := AL;
    end;
  end;

{-------------------------------------------------

    __AlterControlWordBit(op,bit) function
```

```
      alter the state of the indicated Control Word bit.
If op is 'S' or '+', the bit is set;  if op is 'C' or
'-', the bit is cleared.

    returns error code

 ------------------------------------------------------}

function __AlterControlWordBit(op: Char;bit: Word):
Integer;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    if Er=0 then begin
      if upcase(op) in ['S','+'] then ctrl := ctrl or
__BITVALUE[bit]
        else if upcase(op) in ['C','-'] then
          ctrl := ctrl and (__ALLBITS-__BITVALUE[bit]);
      Er := __SetControlWord(ctrl);
    end;
    __AlterControlWordBit := Er;
  end;
{------------------------------------------------------

    __SetAutoShutterMode(state) function
      set the state ('S'or '+' to set, 'C' or '-' to
clear) of the Auto-Shutter Mode.

    returns error code

 ------------------------------------------------------}

function __SetAutoShutterMode(op: Char): Integer;
  begin
    __SetAutoShutterMode := __AlterControlWordBit(op,0);
```

```
  end;

{-------------------------------------------------------

    __SetPosAckMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the Position-Acknowledge Mode.

    returns error code

 -------------------------------------------------------}

function __SetPosAckMode(op: Char): Integer;
  begin
    __SetPosAckMode := __AlterControlWordBit(op,1);
  end;

{-------------------------------------------------------

    __SetBlockMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the Block Mode where the 07 commands (below)
are active.

    returns error code

 -------------------------------------------------------}

function __SetBlockMode(op: Char): Integer;
  begin
    __SetBlockMode := __AlterControlWordBit(op,2);
  end;

{-------------------------------------------------------

    __SetTurnOffDelay(ticks) function
```

```
      sets the idle state's laser turn-off delay to
ticks interrupt counts.

    returns error code

-------------------------------------------------------}

function __SetTurnOffDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0604;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOffDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __SetTurnOnDelay(ticks) function
      sets the idle state's laser turn-on delay to ticks
interrupt counts.

    returns error code

-------------------------------------------------------}

function __SetTurnOnDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0605;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOnDelay := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __SetHomePosition(X,Y) function
      sets the idle state's laser home position.

    returns error code
-------------------------------------------------------}

function __SetHomePosition(X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0606;
      BX := X;
      CX := Y;
      Intr(_LASERVECTOR,__IntReg);
      __SetHomePosition := AL;
    end;
  end;

{-------------------------------------------------------
```

__Execute function

set the background task running. The background task will process those commands added to the command queue from the last Clear Operation up to this __Execute call. For the first __Execute, then background task will start at the beginning of the queue. New commands can be added to the command queue while the background task is running.

When the background task is finished processing the current command block, it will automatically shut down, placing the laser beam at the preset home position.

To make programs written for earlier versions (where the background task free-ran) compatible, __Execute, __ExecuteClear, and __Clear calls (see below) should be used wherever appropriate.

```
    returns error code

 ------------------------------------------------------}

function __Execute: Integer;
  begin
    with __IntReg do begin
      AX := $0700;
      Intr(_LASERVECTOR,__IntReg);
      __Execute := AL;
   end;
  end;

{------------------------------------------------------

    __ExecuteClear function
      perform the equivalent of the __Execute function
(see above) and a __Clear function (see below) function.
The __ExecuteClear function provides a convenient
shorthand for:

        Er := __Execute;
        Er := __Clear;

    returns error code

 ------------------------------------------------------}

function __ExecuteClear: Integer;
  begin
    with __IntReg do begin
      AX := $0701;
      Intr(_LASERVECTOR,__IntReg);
      __ExecuteClear := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __Clear function
      clear the command queue.  Future __Execute and
__ExecuteClear calls will start the background task
processing command queue entries added after the call to
this routine.  An Execute Operation asked for
immediately after a Clear Operation has no effect.
       If __Clear is not called between consecutive
Execute Operations, then background task will reprocess
the command queue entries for the first Execute
Operation.  Sometimes this is desired, and sometimes it
isn't.

    returns error code

 -------------------------------------------------------}

function __Clear: Integer;
  begin
    with __IntReg do begin
      AX := $0702;
      Intr(_LASERVECTOR,__IntReg);
      __Clear := AL;
    end;
  end;

{-------------------------------------------------------

    __SetStepPeriods function
      set the step periods for the multiple passes.
array contains number of step periods to define (1 to
10) and the step period values in 0.82 us (1.19 Mhz)
units.

    returns error code
```

```
 ------------------------------------------------------}

function __SetStepPeriods(var sp_array:
_StepPeriodsType): Integer;
  begin
    with __IntReg do begin
      AX := $0800;
      ES := Seg(sp_array);     { point ES:BX to step
                               periods array }
      BX := Ofs(sp_array);
      Intr(_LASERVECTOR,__IntReg);
      __SetStepPeriods := AL;
    end;
  end;

{------------------------------------------------------

    __SetRedrawCount function
      set the number of passes for the background task's
redraw (1-8).

    returns error code

------------------------------------------------------}

function __SetRedrawCount(count: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0801;
      if count<1 then count := 1
        else if count>10 then count := 10;
      BX := count;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawCount := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __SetRedrawDelay function
      set the number of timer ticks the background task
should delay before starting a redraw pass.

    returns error code

-------------------------------------------------------}

function __SetRedrawDelay(delay: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0802;
      BX := delay;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __Mark function
      marks the current position as the start of the
next redraw.  Usually only necessary at the beginning of
a block.

    returns error code

-------------------------------------------------------}

function __Mark: Integer;
  begin
    with __IntReg do begin
      AX := $0803;
      Intr(_LASERVECTOR,__IntReg);
```

```
      __Mark := AL;
    end;
  end;

{-------------------------------------------------------

    __Redraw function
      causes the background task to perform a multi-pass
redraw operation when it is encountered in the
Background Command Queue.  __Redraw also marks so as to
minimize use of commands.

    returns error code

 -------------------------------------------------------}

function __Redraw: Integer;
  begin
    with __IntReg do begin
      AX := $0804;
      Intr(_LASERVECTOR,__IntReg);
      __Redraw := AL;
    end;
  end;

{-------------------------------------------------------

    __GetTimerCapability function
      returns the multiple timebase status in BX, 0 for
no, 1 for yes.  An error will be generated by old LASER
tasks.

    returns error code

 -------------------------------------------------------}

function __GetTimerCapability: Boolean;
```

```
begin
  with __IntReg do begin
    AX := $0900;
    Intr(_LASERVECTOR,__IntReg);
    if (AL = 0) and (BL = 1) then __GetTimerCapability
:= true
      else __GetTimerCapability := false;
  end;
end;

{-----------------------------------------------------

    __SelectTimerChannel function
      adds a new command to the laser controller
background command queue.  The timer channel to be
selected when the background task reaches this point in
the queue is passed as a parameter.

    returns error code

 -----------------------------------------------------}

function __SelectTimerChannel(Channel: Byte): Integer;
begin
  with __IntReg do begin
    AX := $0901;
    BX := Channel;
    Intr(_LASERVECTOR,__IntReg);
    __SelectTimerChannel := AL;
  end;
end;

{-----------------------------------------------------

    __GetChannelTimebase function
      For the specified channel, the timebase (# of 1.19
MHz ticks) previously set is returned.
```

```
    returns error code

 ------------------------------------------------------}

function __GetChannelTimebase(Channel: Byte;var Ticks:
Word): Integer;
begin
  with __IntReg do begin
    AX := $0902;
    BX := Channel;
    Intr(_LASERVECTOR,__IntReg);
    Ticks := BX;
    __GetChannelTimebase := AL;
  end;
end;

{------------------------------------------------------

    __GetCurrentTimebase function
      the timebase (# of 1.19 MHz ticks) for the current
timer channel is returned.  This is the REAL-TIME tick
count, and it may not agree with previously sent
__AddChangeTimebase commands.

    returns error code

 ------------------------------------------------------}

function __GetCurrentTimebase(var Ticks: Word): Integer;
begin
  with __IntReg do begin
    AX := $0903;
    Intr(_LASERVECTOR,__IntReg);
    Ticks := BX;
    __GetCurrentTimebase := AL;
  end;
```

```
end;

end.^Z
```

```
{

   LibNC
   Library of No-Math Coprocessor Routines
   3D Systems Stereolithography System

   for use in all versions of STEREO and BUILD
      * STEREOC       * BUILDC
      * STEREONC      * BUILDNC

}

unit LibNC;

{$R-,S-,I-,D-,T-,F-,V-,B-,L+ }
{$M 16384,0,655360 }

{$N-}
interface

function LibRound(num : Real) : LongInt;
function LibTrunc(num : Real) : LongInt;

implementation

function LibRound(num : Real) : LongInt;
begin
  LibRound := Round(num);
end;

function LibTrunc(num : Real) : LongInt;
begin
  LibTrunc := Trunc(num);
end;

end.^Z
```

```
{=========================================================
   3D Systems Stereolithography System Software

=========================================================

   MONITOR.PAS
   SLA-1 Laser Beam Monitoring Program


---------------------------------------------------------
   Compile with Turbo Pascal Version 4.0

---------------------------------------------------------

   Recent History:

    3/14/88  Ver 2.80  first release
    4/09/88  Ver 3.00  production release
    4/27/88  Ver 3.05  laser beam movement slew rate
limiting implemented
    6/17/88  Ver 3.10  use the same beam power algorithm
as beam.pas
    7/18/88  Ver 3.20  update VersionId
---------------------------------------------------------}

{$R-,S-,I-,D+,T-,F-,V-,B-,N-,L- }
{$M 16384,0,0 }

program Monitor;

uses Crt,Dos;

type
  str6  = string[6];
  str10 = string[10];
  str16 = string[16];
  BeamProfileVarSensorType = record
```

```
    Xpos              : LongInt;    { inputs }
    Ypos              : LongInt;
    ScanStep          : Integer;
    BkgndThreshold    : Integer;
    InitialDelay      : Integer;
    RowToRowDelay     : Integer;
    ColToColDelay     : Integer;
    XYweight          : LongInt;    { outputs }
    Xweight           : LongInt;
    Yweight           : LongInt;
    TotalWeight       : LongInt;
    HiPoints          : Word;
    SaturationCount   : Integer;
    MaxSensorReading: Integer;
    BeamMap           : array[0..30,0..30] of Integer;
    RowWeight         : array[0..30] of Word;
    ColWeight         : array[0..30] of Word;
  end;
  BeamProfileVarType = array[1..3] of
BeamProfileVarSensorType;
  FindBeamVarSensorType = record
    Xval              : LongInt;    { inputs & updated }
    Yval              : LongInt;
    SpiralDelay       : Word;       { inputs }
    SpiralSideMax     : Integer;
    SpiralTries       : Integer;
    ScanStep          : Integer;
    BkgndThreshold    : Integer;
    NumCenterPasses : Integer;
    Xavg              : Real;       { outputs }
    Yavg              : Real;
  end;
  FindBeamVarType = array[1..3] of
FindBeamVarSensorType;
  FindMethodType = (none,NormalFind,Verify2MilHole);

const
```

```
  VersionId = '3.20';
  ParamFileName = 'BEAM.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  HomeDirectory = 'C:\3DSYS\';
  PowerHistoryFileName   = 'C:\HISTORY\POWER.MON';
  ProfileHistoryFileName = 'C:\HISTORY\PROFILE.MON';
  Blank = '                                    ';
  ColMax = 22;
  RowMax = 22;

  LaserAddr    = $300;
  NumSensors   = 2;
  Timeout = 60000;

var
  FirstVar: Byte;  { must be first variable defined }
  { beam arrays }
  Resistance,Temperature: array[0..100] of Real;
  { sensor arrays }
  SensorChannel,XSensorPos,YSensorPos,SensorBkgnd:
array[1..3] of Word;
  Power,Bk: array[1..3] of Real;
  Xsp,Ysp: array[1..3] of Real;
  SensorCalib: array[1..3,1..4] of Real;
  LastField: array[1..4] of Integer;
  { miscellaneous }
  key: Char;
  Transparent,BeamFound: Boolean;
  CurrentDate,CurrentTime,OutFileName,SingleBar,CmdLine:
String;
  XRest,YRest,oldX,oldY,TempChannel,TempCalPts: Word;
  i,PeakReading,Sensor,Select,SensorNum: Integer;
  GoodCount,Sec,Minu,Hour,Day,WeekDay,Mon,Year: Integer;
  Xpos,Ypos: LongInt;
  Xaverage,Yaverage,TotWtAvg: Real;
  OldExitProc: Pointer;
  BeamProfileVariables: BeamProfileVarType;
```

```
  FindBeamVariables: FindBeamVarType;
  CalibDelays: array[1..3,1..3] of LongInt;
ErrorFile,ParamFile,PowerHistoryFile,ProfileHistory
File: Text;
  SlewLimitMaxStep,SlewLimitDelay: Word;
  LastVar: Byte;  { should be last variable defined }


{-------------------------------------------------------

                              Utility Routines

-------------------------------------------------------}

procedure beep;
begin
  Sound(1000);
  delay(300);
  NoSound;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
```

```
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);

  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
  Str(num,temp);
  LongStr := temp;
end;

function RealStr(num: Real;AfterDecimal: Integer):
String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal,temp);
```

```
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc := copy(blank,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
```

```
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;

function WordVal(line: String;var code: Integer): Word;
var
  code2: Integer;
  num: Word;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
var
  ofs,code2: Integer;
  num: Real;
begin
  ofs := 0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
```

```
  end;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  RealVal := num;
end;

{
  Show error and store it on disk
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'MONITOR: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{$F+}
procedure ExitMonitor;
{$F-}
begin
  ExitProc := OldExitProc;
```

```
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
```

```
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;    { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;    { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;    { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $09;    { get year }
                Year := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then good :=
true;
              end;
            end;
          end;
        end;
      end;
    end;
    Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
```

```
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetTimeStamp;
var
  tday,tmon,tyear,tsec,tminu,thour: str10;
  AmPm: Char;
begin
  GetTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  CurrentDate := tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
```

```
end;

function TimeStamp: String;
begin
  GetTimeStamp;
  TimeStamp := CurrentDate + ' ' + CurrentTime;
end;

function ReadLaserBoard(Channel: Integer): Integer;
var
  count: LongInt;
  ADC,i: Integer;
begin
  count := Timeout;
  port[LaserAddr+12] := Channel;
  { must wait 10 us for EOS to go LO (286 @ 4us/loop) }
  i := 1;
  while i < 10 do inc(i);
  repeat   { then approx 110 us for conversion with
signal D7 HI }
    dec(count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if PeakReading < ADC then PeakReading := ADC;
  if count = 0 then ReadLaserBoard := -1
    else ReadLaserBoard := ADC;
end;

procedure ReadTemperature(var Resist,Temper: Real);
var
  temp: Real;
  It,ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board
  use 1/T, LOG(R) function TO linearize the
interpolation
```

```
}
  ADC := ReadLaserBoard(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC);    { from
20K resistor on JB Board }
  if ADC>214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] > Resist)
do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It]));
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) - ln(Resist)));
  end;
  Temper := Temper - 273.15;
end;

procedure println(text: String);
begin
  if not transparent then writeln(text);
end;


{-------------------------------------------------------

                          BEAM SCANNING ROUTINES

-------------------------------------------------------


                                  ************
                                  * MoveBeam *
```

```
                                        ************

  Routine to move mirrors to X,Y directly
  Note: X,Y can be any kind of variable, we change to
integers and get ADC% (for speed: later fixed addrs
routines for each)
}
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 then pos := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
 { x is bit 0, y is bit 1 }
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInXAxis(x: LongInt);
var
  stepX: LongInt;
begin
  if abs(x - oldX) > SlewLimitMaxStep then begin
    if x - oldX > 0 then begin
      stepX := oldX + SlewLimitMaxStep;
      while stepX < x do begin
        AxisMove(stepX,1);
        inc(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepX := oldX - SlewLimitMaxStep;
      while stepX > x do begin
        AxisMove(stepX,1);
        dec(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end;
```

```
  end;
  AxisMove(x,1);
  oldX := x;
end;

procedure MoveInYAxis(y: LongInt);
var
  stepY: LongInt;
begin
  if abs(y - oldY) > SlewLimitMaxStep then begin
    if y - oldY > 0 then begin
      stepY := oldY + SlewLimitMaxStep;
      while stepY < y do begin
        AxisMove(stepY,2);
        inc(stepY,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepY := oldY - SlewLimitMaxStep;
      while stepY > y do begin
        AxisMove(stepY,2);
        dec(stepY,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end;
  end;
  AxisMove(y,2);
  oldY := y;
end;

procedure MoveBeam(x,y: LongInt);
begin
  MoveInXAxis(x);
  MoveInYAxis(y);
end;
```

```
{
                                    ******************
                                    * GetBeamProfile *
                                    ******************

}

procedure SetupBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer;X,Y: Real;_ScanStep,_BkgndThreshold:
Integer);
begin
  with BeamProfileVars[Sensor] do begin
    if X               >= 0 then Xpos           :=
trunc(X);
    if Y               >= 0 then Ypos           :=
trunc(Y);
    if _ScanStep       >= 0 then ScanStep       :=
_ScanStep;
    if _BkgndThreshold >= 0 then BkgndThreshold :=
_BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;

procedure GetBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer);
var
  Pass,row,col,ADC,delay,waste: Integer;
  Xcorner,Ycorner: LongInt;
begin
  with BeamProfileVars[Sensor] do begin
    XYweight        := 0;
```

```
    Xweight          := 0;
    Yweight          := 0;
    TotalWeight      := 0;   { for power, not removing
noise till end }
    SaturationCount := 0;
    HiPoints         := 0;
    BeamFound        := false;
    for col:=1 to ColMax do ColWeight[col] := 0;
    for row:=1 to RowMax do RowWeight[row] := 0;
    Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
    Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
    MoveBeam(Xcorner,Ycorner);
    waste := 0;
    while waste < InitialDelay do inc(waste);
    PeakReading := 0;
    for col:=1 to ColMax do begin   { prefer to return
around edges after line }
      Xpos := Xcorner + col * ScanStep;
      Ypos := Ycorner;
      MoveBeam(Xpos,Ypos);
      waste := 0;
      while waste < ColToColDelay do inc(waste);
      for row:=1 to RowMax do begin
        inc(Ypos,ScanStep);
        MoveInYAxis(Ypos);
        waste := 0;
        while waste < RowToRowDelay do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        inc(TotalWeight,ADC);
        if ADC < BkgndThreshold then ADC := 0
          else begin
            inc(HiPoints);
            BeamFound := true;
          end;
        if ADC > 210 then inc(SaturationCount);
        inc(ColWeight[col],ADC);
        inc(RowWeight[row],ADC);
```

```
          BeamMap[row,col] := ADC;
          inc(XYweight,ADC);
          inc(Xweight,Xpos * ADC);    { want floating point
                                        numbers here }
          inc(Yweight,Ypos * ADC);
        end; {for row}
      end; {for col}
      MaxSensorReading := PeakReading;
    end;
  end;


  {
                                    ************
                                    * GoodRead *
                                    ************
  }

  procedure GoodRead(Sensor,iter: Integer);
  var
    Try: Integer;
  begin
    with BeamProfileVariables[Sensor] do begin
      Xaverage  := 0;
      Yaverage  := 0;
      TotWtAvg  := 0;
      GoodCount := 0;
      for Try := 1 to iter do begin
        SetupBeamProfile(BeamProfileVariables,Sensor,Xsp
  [Sensor],
          Ysp[Sensor],-1{ScanStep},-1{BkgndThreshold});
        GetBeamProfile(BeamProfileVariables,Sensor);
        if XYweight > 0 then begin
          Xaverage := Xweight / XYweight + Xaverage;
          Yaverage := Yweight / XYweight + Yaverage;
          TotWtAvg := TotWtAvg + TotalWeight -
  SensorBkgnd[Sensor];
```

```
        inc(GoodCount);
      end;
    end; {for Try}
    if GoodCount > 0 then begin
      Xaverage := Xaverage / GoodCount;
      Yaverage := Yaverage / GoodCount;
      TotWtAvg := TotWtAvg / GoodCount;
    end;
    Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
  end;
end;


{-------------------------------------------------------
                                    HARDWARE ROUTINES
-------------------------------------------------------


                                     ***********
                                     * IOSetup *
                                     ***********
}

procedure IOSetup;
begin
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
end;

{
                                    ************
                                    * FindBeam *
                                    ************
}
```

```
procedure SetupFindBeam(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer;X,Y:
Real;_SpiralSideMax,_SpiralTries,_ScanStep,
  _BkgndThreshold,_SpiralDelay,_NumCenterPasses:
Integer);
begin
  with FindBeamVars[Sensor] do begin
    if X                >= 0 then Xval            :=
trunc(X);
    if Y                >= 0 then Yval            :=
trunc(Y);
    if _SpiralSideMax   >= 0 then SpiralSideMax   :=
_SpiralSideMax;
    if _SpiralTries     >= 0 then SpiralTries     :=
_SpiralTries;
    if _ScanStep        >= 0 then ScanStep        :=
_ScanStep;
    if _SpiralDelay     >= 0 then SpiralDelay     :=
_SpiralDelay;
    if _BkgndThreshold  >= 0 then BkgndThreshold  :=
_BkgndThreshold;
    if _NumCenterPasses >= 0 then NumCenterPasses :=
_NumCenterPasses;
  end;
end;

function SpiralSearchForHole(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer): Boolean;
var
  i,ADC,SpiralSide,waste: Integer;
  X,Y: LongInt;
begin
  with FindBeamVars[Sensor] do begin
    X := trunc(Xval);
```

```
    Y := trunc(Yval);
    MoveBeam(X,Y);
    delay(SpiralDelay);
    BeamFound := false;
    SpiralSide := 1;
    while (SpiralSide <= SpiralSideMax) and not
BeamFound do begin
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin

        MoveInXAxis(X);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(X,ScanStep);
        inc(i);
      end;
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInYAxis(Y);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(Y,ScanStep);
        inc(i);
      end;
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := BeamFound;
```

```
  end;
end;

function FindBeam(var FindBeamVars:
FindBeamVarType;Sensor: Integer;
  FindMethod: FindMethodType): Boolean;
const
  HoleSeparation = 600;
var
  i,ADC,pass,CantFind,SpiralSide: Integer;
  XspAv,YspAv: Real;
  X,Y,GoodX,GoodY: LongInt;
  BeamFound: Boolean;
begin
  FindBeam := true;
  with FindBeamVars[Sensor] do begin
    CantFind  := 0;
    repeat
      X := trunc(Xval);
      Y := trunc(Yval);
      MoveBeam(X,Y);      { gets us to best starting
                          position }
{
  now spiral to search for holes
}
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
       -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      BeamFound := SpiralSearchForHole
(FindBeamVariables,Sensor);
      if not BeamFound then begin
        SetupFindBeam(FindBeamVariables,
Sensor,X,Y,SpiralSideMax * 2,
          -1{SpiralTries},-1{ScanStep},BkgndThreshold -
1,-1{SpiralDelay},
```

```
          -1{NumCenterPasses});
        inc(CantFind);
      end;
    until (CantFind >= SpiralTries) or BeamFound;
    if CantFind >= SpiralTries then begin
      MoveBeam(Xval,Yval);
      FindBeam := false;
      exit;
    end;
    X := Xval;
    Y := Yval;
{
  Verify that we have 2 mil hole...

  1/2 mil    1 mil ) left rear            2 mil     4
mil ) right front
                   ) of vat
  ) of vat
   4  mil    2 mil )                      1 mil    1/2
mil )

     Sensor #1                                Sensor #2
```

We have the position of one of the four holes: 1/2, 1, 2, or 4 mils. We want to find the 2 mil hole closest to the vat. Note that the orientation of the holes is different for Sensors 1 and 2.

Hole Size Check Procedure:

1. We may have the position for the 1/2 or 1 mil hole. Move towards the vat in the X-axis and search for the 1 or 2 mil hole. If the beam is seen, adopt the new coordinates.

```
   2.  Now we may have the position for the 4 mil hole.
Move towards the vat in the Y-axis and search for the 2
mil hole.  If the beam is seen, use the new coordinates.
}
    if (Sensor < 3) and (FindMethod = Verify2MilHole)
then begin
      GoodX := X;
      GoodY := Y;
      if X < 32767 then inc(X,HoleSeparation)
        else dec(X,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,
SpiralSideMax div 2,
        -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
          GoodX := Xval;
          GoodY := Yval;
        end;
      X := GoodX;
      Y := GoodY;
      if Y < 32767 then inc(Y,HoleSeparation)
        else dec(Y,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
        -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
      X := GoodX;
      Y := GoodY;
    end;
```

```
    XspAv := 0;
    YspAv := 0;
    if BeamFound then begin
      for pass:=1 to NumCenterPasses do with
BeamProfileVariables[Sensor] do begin
        CantFind := 0;
        { discard the first }
        SetupBeamProfile(BeamProfileVariables,
Sensor,X,Y,
          4{ScanStep},trunc(Bk[Sensor] + 0.5));
        repeat
          GetBeamProfile(BeamProfileVariables,Sensor);
          if XYweight = 0 then inc(CantFind);
        until (XYweight > 0) or (CantFind > 3);
        if CantFind > 3 then begin
          FindBeam := false;
          exit;
        end;
{
  Later select sensor hole size automatically
}
        X := trunc(Xweight / XYweight);   { better
coordinates }
        Y := trunc(Yweight / XYweight);
        Power[Sensor] := (TotalWeight -
SensorBkgnd[Sensor])
          * (SensorCalib[Sensor,3]);
        XspAv := XspAv + Xweight / XYweight;
        YspAv := YspAv + Yweight / XYweight;
      end; {for pass}
      if NumCenterPasses > 1 then begin
        XspAv := XspAv / NumCenterPasses;
        YspAv := YspAv / NumCenterPasses;
      end;
    end; {if BeamFound}
    Xval := X;       { update integer coordinates }
```

```
    Yval := Y;
    Xavg := XspAv;   { update average coordinates }
    Yavg := YspAv;
  end;
end;

function CoarseFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] +
0.5){BkgndThreshold},1{SpiralDelay},
    1{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind) then
begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] +
0.5){BkgndThreshold},100{SpiralDelay},
    2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,Verify2MilHole)
then begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  FineFindBeam := BeamFound;
end;
```

```
function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) then
    BeamFound := FineFindBeam(Sensor);
  FindLaserBeam := BeamFound;
end;


{  -  -  -  -  -  }


function FindHoles2(Sensor: Integer): Boolean;
{
  When we want to use new starting point
   updates hole position
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;

function FindHoles(Sensor: Integer): Boolean;
{
  Updates hole position
  Needs to confirm that we have 2 mil hole
}
var
  BkRead: LongInt;
  ADC,row,col: Integer;
begin
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
  BkRead := 0;
  for col:=1 to ColMax do begin
    for row:=1 to RowMax do begin
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
```

```
      inc(BkRead,ADC);
    end;
  end;
  SensorBkgnd[Sensor] := BkRead;
  { remove noise for centering }
  Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax) + 3;
  Xsp[Sensor] := XSensorPos[Sensor];   { will be best
starting point }
  Ysp[Sensor] := YSensorPos[Sensor];
  FindHoles := FindHoles2(Sensor);
end;


{=======================================================}


{
  Read BEAM.PRM parameters file
}
procedure ReadBeamParamsFile;
var
  ParamFile: text;
  i,count,code,code1,code2,Sensor,Pinhole: Integer;
  cmd,ParamLine: String;

  procedure GetParamFileLine;
  begin
    if eof(ParamFile) then ParamLine := ''
    else repeat
      readln(ParamFile,ParamLine);
    until eof(ParamFile) or ((ParamLine <> '') and
(ParamLine[1] <> '!'));
  end;

begin
  {$I-}
  assign(ParamFile,ParamFileName);
```

```
  reset(ParamFile);
  if IOresult<>0 then begin
    assign(ParamFile,HomeDirectory+ParamFileName);
    reset(ParamFile);
    if IOresult<>0 then halt(3);
  end;
  {$I+}
  GetParamFileLine;
  if copy(ParamLine,1,1) = chr(34) then begin   { handle
quoted options line }
    i := pos(chr(34),copy(ParamLine,2,255));
    if i>0 then ParamLine[0] := chr(i);
    ParamLine := NoQuotes(ParamLine);
    GetParamFileLine;
  end;
  XRest := WordVal(ParamLine,code);
  YRest := WordVal(copy(ParamLine,code+1,255),code);
  GetParamFileLine;
  SensorNum := WordVal(ParamLine,code);
  for Sensor:=1 to SensorNum do begin
    GetParamFileLine;
    SensorChannel[Sensor] := WordVal(ParamLine,code);
    GetParamFileLine;
    XSensorPos[Sensor] := WordVal(ParamLine,code);
    YSensorPos[Sensor] :=
WordVal(copy(ParamLine,code+1,255),code);
    for Pinhole:=1 to 4 do begin
      GetParamFileLine;
      SensorCalib[Sensor,Pinhole] :=
RealVal(ParamLine,code);
    end;
  end; {for Sensor}
  GetParamFileLine;
  TempChannel := WordVal(ParamLine,code);   { assume
only one thermometer at first }
  GetParamFileLine;
  TempCalPts := WordVal(ParamLine,code);
```

```
  for i:=1 to TempCalPts do begin
    { Resistance[i],Temperature[i] may not be integers }

    GetParamFileLine;
    Resistance [i] := RealVal(ParamLine,code);
    Temperature[i] :=
RealVal(copy(ParamLine,code+1,255),code) + 273.15;
  end;
  for i:=1 to 3 do begin   { read in Fast, Medium, &
Slow Calib Delay values }
    GetParamFileLine;
    CalibDelays[i,1] := LongVal(ParamLine,code1);
    CalibDelays[i,2] :=
trunc(RealVal(copy(ParamLine,code1+1,255),code2)*10+0.5)
;
    CalibDelays[i,3] :=
trunc(RealVal(copy(ParamLine,code2+code1+1,255),code)*10
+0.5);
  end;
  SlewLimitMaxStep := 1000;
  SlewLimitDelay   := 2;
  if not eof(ParamFile) then begin
    GetParamFileLine;
    SlewLimitMaxStep := WordVal(ParamLine,code);
    if SlewLimitMaxStep < 10 then SlewLimitMaxStep :=
10;
    GetParamFileLine;
    SlewLimitDelay   := WordVal(ParamLine,code);
  end;
  close(ParamFile);
  Resistance [0] := Resistance[1]  * 2 - Resistance [2];
  Temperature[0] := Temperature[1] * 2 - Temperature[2];
end;


{-----------------------------------------------------}
```

```
procedure SetLaser;
begin
  IOSetup;
  for i:=1 to 3 do begin
    Xsp[i] := XSensorPos[i];
    Ysp[i] := YSensorPos[i];
  end;
  oldX := XRest;   { initialize beam position }
  oldY := YRest;
  AxisMove(oldX,1); { first time is not slew rate
                      limited }
  AxisMove(oldY,1);
  delay(500);
end;

procedure MonitorLaserBeamPower;
begin
  println('Monitoring Laser Beam Power...');
  SetLaser;
  assign(PowerHistoryFile,PowerHistoryFileName);
  {$I-}
  append(PowerHistoryFile);
  if IOresult <> 0 then rewrite(PowerHistoryFile);
  {$I+}
  if IOresult = 0 then begin
    for Sensor := 1 to NumSensors do begin
      if not FindHoles(Sensor) then begin
        beep;
        beep;
        Power[Sensor] := 0;
      end else GoodRead(Sensor,10);
    end;
    writeln(PowerHistoryFile,TimeStamp + '  Sensor
#1=',RealStr(Power[1],2),
      '  Sensor #2=',RealStr(Power[2],2));
    close(PowerHistoryFile);
  end else beep;
```

```
end;

procedure MonitorLaserBeamProfile;
var
  row,col: Integer;
begin
  println('Monitoring Laser Beam Profile...');
  SetLaser;
  assign(ProfileHistoryFile,ProfileHistoryFileName);
  {$I-}
  append(ProfileHistoryFile);
  if IOresult <> 0 then rewrite(ProfileHistoryFile);
  {$I+}
  if IOresult = 0 then begin
    for Sensor := 1 to NumSensors do begin
      if FindHoles(Sensor) then begin
        writeln(ProfileHistoryFile,TimeStamp,' Sensor
#',IntStr(Sensor),':');
        for col:=1 to ColMax do begin
          write(ProfileHistoryFile,' ');
          for row:=1 to RowMax do begin
            write(ProfileHistoryFile, BeamProfile
Variables[Sensor].BeamMap[row,col]:3);
          end; {for row}
          writeln(ProfileHistoryFile);
        end; {for col}
      end else begin
        beep;
        beep;
        writeln(ProfileHistoryFile,TimeStamp,' Sensor
#',IntStr(Sensor),
          ' beam not found');
      end;
    end;
    close(ProfileHistoryFile);
  end else beep;
end;
```

```
procedure ShowTitle;
begin
  TextMode(C80);
  SingleBar := many(#196,45);
  writeln(SingleBar);
  writeln('  MONITOR, Laser Beam Monitor Program');
  writeln('  Version '+VersionId);
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography
System');
  writeln(SingleBar);
end;


{ - - -  M A I N  P R O G R A M - - - }


begin  { Monitor }
  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);
{ clear variables }
  OldExitProc := ExitProc;
  ExitProc := @ExitMonitor;
  CheckBreak := true;
  OutFileName := '';
  ReadBeamParamsFile;
  ReadCommandLine(CmdLine);
  CmdLine := upper(CmdLine);
  i := pos ('/T',CmdLine);
  if i > 0 then begin
    Transparent := true;
    delete(CmdLine,i,2);
  end;
  if not Transparent then begin
    ShowTitle;
    writeln;
    writeln;
```

```
  end;
  if pos('POWER',CmdLine)    > 0 then
MonitorLaserBeamPower;
  if pos('PROFILE',CmdLine) > 0 then
MonitorLaserBeamProfile;
  QuietSystemError('Success.');
end.
^Z
```

```
Program SLAGRAPH;
{
                        3D Systems SLA1 Graphics Software
                        ---------------------------------
                        Copyright (C) 1988 3D Systems Inc.


  Change Log:
    2/15/88  Test260  Initial Pascal version
    2/18/88    2.50   Adjust 3D graph scaleing
                      Change layer select of graph for
                      PgUp and PgDn
                      Correct ASC Error handling
    2/19/88    2.60   Load Graphic drivers into EXE
               2.61   Change command lines to () format
                      Accept either ENDSOLID or END
                      SOLID in ASCII files
                      Remove ref. to ARIES
    2/26/88    2.62   Correct FILE to LAYER in slice
                      file menu
    3/02/88    2.70   Change name from SLAGRAPH to GRAPH
                      Correct spelling (Graphis to
                      Graphics)
                      Read last line of text file even
                      if no CR
                      Rotation for triangles -360 to 360
                      RHR for axis rotation
                      Add Axis on triangle screen
    4/06/88     3.00  Production Release, change menus
                      and support axis switch
    4/29/88     3.10  Add pan, zoom, delay, Xtended
                      viewing mode, Surface mode,
                      Normal check mode, and more.
    6/22/88     3.11  Change pan from pixel input to -10
                      to +10 input
    6/27/88     3.12  Support EGA, VGA, and Hercules
```

```
                        displays, add rectangles
      7/18/88     3.20  Update Version
      7/25/88     3.21  In surface mode, ignore all facets
                        whose normals are <= 0.
      8/12/88     3.22  Add VGA color support
      8/17/88     3.23  Change main menu #3, and center
                        all options
      8/26/88     3.24  Allow '.5' string by using
                        function RealVal, correct
                        begin/end facet setting, and
                        correct pausing if a
                        range is selected.
      9/09/88     3.25  fix totaltriangles in select range
                        pause/quit, and rot.
      9/14/88     3.26  make number of facets (max) =
                        total triangles, not tt-1
}
{$R-,S-,I+,D-,T-,F-,V-,B-,N-,L- }
{$M 16384,0,655360 }

  uses
    Crt,
    Dos,
    Graph,
    Drivers;

  const

    VERSION : String = '3.26';
    MAX_WORD : Word = 65535;
    MAX_REAL : Real = 1.7E37;
    MIN_REAL : Real = -1.7E37;
    ZERO_DEF = 1E-5;
    ESC      : Char = #27;
    CR       : Char = #13;
    LF       : Char = #10;
    EOF_CHAR : Char = #26;
```

```
    BUF_SIZE = 8191;
    Spaces : String =        '                          ';
    MAX_PTR_ARRAY = 16382;

    { Slice Constants }
    MAX_LAYERS    = 16382;
    LAYER_SIZE    = 16;
    MAX_SECTIONS  = 5;
    SliceBlocks : Array [0..4] of String[3] =
('L','FU','FD','NFU','NFD');
    L_SECTIONS    = $0007;
    L_SECTION     = $8000;
    LB_SECTION    = $0001;
    LH_SECTION    = $0002;
    FU_SECTIONS   = $0038;
    FD_SECTIONS   = $01C0;
    NFU_SECTIONS  = $0E00;
    NFD_SECTIONS  = $7000;
    F_SECTIONS    = $01F8;
    NF_SECTIONS   = $7E00;

    { Triangle Constants }
    MAX_TRIANGLES  = 10921;
    TRIANGLE_SIZE  = 10;
    ASC_FILE       = 1;       { File Type Constants }
    BINARY_FILE    = 2;
    ARIES_FILE     = 3;
    COOR_CHAR : Array [1..3] of Char = ('X','Y','Z');

  type
    DynamicSLWordArray     = Array [0..16382] of Word;
    DynamicSLLongIntArray  = Array [0..16382] of
LongInt;
    DynamicTRILongIntArray = Array [0..10921] of
LongInt;
    DynamicTRIRealArray    = Array [0..6552] of Real;
    PointerStruct          = Array [0..255] of Byte;
```

```
    PointerType             = ^PointerStruct;

  var
    ExitSave                                : Pointer;
    HeapTop                                 : Pointer;
    InBuffer                                : Array
[0..BUF_SIZE] of Char;
    InBufStr                                : String;
    InBufEnd                                : Integer;
    InBufPtr                                : Integer;
    CurrFilePos                             : LongInt;
    StartLinePos                            : LongInt;
    LinePosValid                            : Boolean;
    CurrLinePos                             : LongInt;
    EndOfFile                               : Boolean;
    FileError, SSorted, WSorted             : Boolean;
    ErrLine                                 : String;
    ch, DisplayMode                         : Char;
    rc                                      : Integer;
    iw,jw                                   : Word;
    TempInt                                 : Integer;
    TempByte                                : Byte;
    TempWord                                : Word;
    TempBoolean                             : Boolean;
    TempReal                                : Real;
    TempLong                                : LongInt;
    TempStr,TempStr1                        : String;
    TempChar,DisplayKey                     : Char;
    InputLine                               : String;
    MonoDisplay, StopDisplayLoop            : Boolean;
    GraphDriver, GraphMode                  : Integer;
    SizeX,SizeY                             : Integer;
    AspectRatio                             : Real;
    GraphScale                              : Real;
    GraphCenterX,GraphCenterY               : Word;
    GrOffsetX,GrOffsetY,GrOffsetZ           : Real;
```

```
    GF2D                                : Array
[1..3,1..2] of Integer;
    DX,DY,DisplayDelay,HoriOffset,VertOffset      :
integer;
    InputStr                            : String;

InputInt,ValCode,BeginFacet,EndFacet,NormLength,Xpixels,
Ypixels : integer;
    zoom,barN                           : real;
    Vector3,Vector1,SliceNormal,DeltaNormal       :
array[1..3] of real;
    NormalsAreOK                         : boolean;
{   gftest                              : text; }
    regs                                : registers;
    { Triangle FIle Global Vars }
    TriangleFile                        : File;
    TriangleFilename                    : String;
{ Filename of Triangle File  }
    TriangleFiletype                    : Byte;
{ File Type of Input File }
    TrianglePtr                         :
^DynamicTRILongIntArray; { Pointers to Triangle File }
    TriangleAxisValue                   :
^DynamicTRIRealArray;    { Z Values of each triangle }
    TriangleNormal                      : Array [1..3] of
Real;
    TriangleVertex                      : Array
[1..3,1..3] of Real;
    SavePastVertex                      : Array
[1..3,1..3] of Real;
    TriangleAttrib                      : Word;
    TriangleTotal                       : LongInt;
    TriangleFileMin                     : Array [1..3] of
Real;
    TriangleFileMax                     : Array [1..3] of
Real;
    TriangleSelected                    : LongInt;
```

```
    TriangleSelectStart                : Real;
    TriangleSelectEnd                  : Real;
    TriangleSelectMin                  : Array [1..3] of
Real;
    TriangleSelectMax                  : Array [1..3] of
Real;
    TriangleWindowingFile              : Boolean;
    TriangleAngle                      : Array [1..3] of
Integer;
    TriangleAxis                       : Byte;
    Gr3DMat                            : Array
[0..2,0..2] of Real;
    Total_Triangles                    : LongInt;

    { Slice File Global Vars }
    SliceFile                          : File;
    SliceFilename                      : String;
{ Filename of Slice File  }
    SliceFileXMin,SliceFileXMax        : Word;
{ File Max and Min values }
    SliceFileYMin,SliceFileYMax        : Word;
    SliceLastLayer                     : Integer;
{ Pointer to last layer   }
    SliceLayerValue                    :
^DynamicSLWordArray;    { Layer Values }
    SliceLayerPtr                      :
^DynamicSLLongIntArray; { Pointer to Layer in file }
    SliceLayerBlocks                   :
^DynamicSLWordArray;    { Layer Blocks }
    SliceLayerXMin,SliceLayerXMax      :
^DynamicSLWordArray;    { Layer Max and Min values }
    SliceLayerYMin,SliceLayerYMax      :
^DynamicSLWordArray;
    SliceLayer                         : Word;
{ Current Layer }
    SliceWindowingFile                 : Boolean;
    SliceOutputBlocks                  : Word;
```

```
    Total_Layers                          : LongInt;

{$I view.inc}

{-------------------------------------------------------
  |                        Start of Main Triangle Routines
                    |
-------------------------------------------------------}

function ReadTriangle : Boolean;
  { Read in one triangles data from input files current
location }
  var
    ParsingLine : String;
    i,j     : Word;
  begin
    if TriangleFiletype=ASC_FILE then begin
      TriangleAttrib:=0;
      ReadLine(TriangleFile); { FACET NORMAL i j k }
      TempStr:=ParseLine(InputLine);
      if Copy(TempStr,1,3)='END' then EndOfFile:=TRUE
      else begin
        if (TempStr<>'FACET') OR
(ParseLine(InputLine)<>'NORMAL') then FileError:=TRUE;
        if Not FileError then for i:= 1 to 3 do begin
          Val(ParseLine(InputLine),TriangleNormal
[i],rc);
          if rc<>0 then FileError:=TRUE;
        end;
        ReadLine(TriangleFile); { OUTER LOOP }
        if (ParseLine(InputLine)<>'OUTER') OR
(ParseLine(InputLine)<>'LOOP') then FileError:=TRUE;
        if Not FileError then for i:= 1 to 3 do begin
          ReadLine(TriangleFile); { VERTEX }
          if ParseLine(InputLine)<>'VERTEX' then
FileError:=TRUE
          else for j:= 1 to 3 do begin
```

```
            Val(ParseLine(InputLine),TriangleVertex
[i,j],rc);
            if rc<>0 then FileError:=TRUE;
          end;
        end;
      end;
    end else begin
      { A Binary File }
      for i:=0 to 2 do
ConvertStoR(@InBuffer[InBufPtr+i*4],@TriangleNormal
[i+1]);
      if InBufEnd-InBufPtr>11 then for i:=0 to 2 do
        for j:=0 to 2 do ConvertStoR(@InBuffer[InBufPtr+
12+i*12+j*4],@TriangleVertex[i+1,j+1]);
    end;
    ReadTriangle:=Not (FileError or EndOfFile);
  end;

procedure GraphTriangles(var ch:Char);
  { Graph the selected triangles to screen }
  const
     XRedBox : array[1..4] of PointType = ((x:  1;
y:339),
                                           (x:  1;
y:286),
                                           (x:280;
y:286),
                                           (x:280;
y:339));
     RedBox : array[1..4] of PointType =  ((x:  1;
y:350),
                                           (x:  1;
y:338),
                                           (x:280;
y:338),
                                           (x:280;
y:350));
```

```
  var
    CurrentColor, savePastColor, i,j  : integer;
    index : LongInt;
    min,max : Real;
    MidPoint: array [1..2,1..3] of real;
    savenormx,savenormy: Integer;
  begin
    min:=MAX_REAL;
    max:=MIN_REAL;
    Init3D;
    if TriangleWindowingFile then
      SetupGraphicR(TriangleFileMin[1],
TriangleFileMax[1],
                    TriangleFileMin[2],
TriangleFileMax[2],
                    TriangleFileMin[3],
TriangleFileMax[3])
    else
      SetupGraphicR(TriangleSelectMin[1],
TriangleSelectMax[1],
                    TriangleSelectMin[2],
TriangleSelectMax[2],
                    TriangleSelectMin[3],
TriangleSelectMax[3]);
    GrTextXY('Triangle Filename   :
'+TriangleFilename,1,1);
    Str(TriangleSelected,TempStr);
    GrTextXY('Triangle Count      : '+TempStr,1,2);
    GrTextXY('or <esc> for main menu',43,3);
    GrTextXY('Q to quit current display-',43,2);
    GrTextXY('press any key to pause-',43,1);

    index := beginFacet;

    DisplayKey := 'F'; { Step, Forward, Run, Stop=
space, Back, Quit }
    StopDisplayLoop:=false;
```

```
    SavePastColor := 0;
    CurrentColor := 7; {lightgray}
    SelectGraphColor(CurrentColor);
    DrawAxis;

    while DisplayKey<>'Q' do begin
     SelectGraphColor(red);
     SetLineStyle(SolidLn,$00,NormWidth);
     SetFillStyle(SolidFill,red);
     FillPoly(4,RedBox);
     if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
     GrTextXY(' Drawing in Process... ',1,35);
    repeat
       if ((TriangleAxisValue^[index-1]>=
TriangleSelectStart) and
         (TriangleAxisValue^[index-1]<=
TriangleSelectEnd))
         or (displaymode='S') then begin
         PositionFile(TriangleFile,TrianglePtr^
[index-1]);
        TempBoolean:=ReadTriangle;
        if abs(TriangleNormal[TriangleAxis])>1-ZERO_DEF
then
          if TriangleNormal[TriangleAxis]>0 then
CurrentColor:=10
              { LightGreen represents FU }
          else CurrentColor:=9
              { LightBlue represents FD}
        else if TriangleNormal[TriangleAxis]>0 then
CurrentColor:=2
              { Green represents U}
             else CurrentColor:=1;
             { Blue represents D }
        if DisplayMode = 'X' then begin
           if (SavePastColor <> 0) then begin
```

```
                { the triangle is first drawn in red, now
it is redrawn in
                  its proper current color for the final
display - noncurrent }
                SelectGraphColor(SavePastColor);
                DrawPartLine(savepastVertex[1,1],
savepastVertex[1,2],savepastVertex[1,3],
                        savepastVertex[2,1],savepastVertex
[2,2],savepastVertex[2,3]);
                DrawPartLine(savepastVertex[2,1],
savepastVertex[2,2],savepastVertex[2,3],
                        savepastVertex[3,1],savepastVertex
[3,2],savepastVertex[3,3]);
                DrawPartLine(savepastVertex[3,1],
savepastVertex[3,2],savepastVertex[3,3],
                        savepastVertex[1,1],savepastVertex
[1,2],savepastVertex[1,3]);
                { redraw the normal in grey instead of
yellow }
                SelectGraphColor(LightGray);
                DrawPartLine(MidPoint[1,1],MidPoint
[1,2],MidPoint[1,3],
                        MidPoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
                OutTextXY(saveNormX,saveNormY,'+');
            end;
            for i:=1 to 3 do begin
                for j:=1 to 3 do savePastVertex[i,j]:=
TriangleVertex[i,j];
                MidPoint[1,i]:=(TriangleVertex[1,i]
+TriangleVertex[2,i]+
                                TriangleVertex[3,i])/3.0;
            end;
            SavePastColor:=CurrentColor;
            CurrentColor := 4; {red}
            SelectGraphColor(CurrentColor);
```

```
            Vertex3DZoom(MidPoint[1,1],MidPoint[1,2],
        MidPoint[1,3],
                                saveNormX,saveNormY);
            saveNormX:=saveNormX-3;
            saveNormY:=saveNormY-3;
            SelectGraphColor(White);
            OutTextXY(saveNormX,saveNormY,'+');
            SelectGraphColor(14); { yellow }
            for i:= 1 to 3 do MidPoint[2,i]:=
        MidPoint[1,i]+
                  (TriangleNormal[i]*(NormLength/
        100)/Zoom);
            DrawPartLine(Midpoint[1,1],MidPoint
        [1,2],MidPoint[1,3],
                                Midpoint[2,1],MidPoint[2,2],
        MidPoint[2,3]);
            SelectGraphColor(red);
            SetFillStyle(SolidFill,red);
            FillPoly(4,XRedBox);
            if MonoDisplay then SetColor(Black) else
        SelectGraphColor(white);
            InputInt := index;
            Str(InputInt:6,TempStr);
            GRTextXY(' Facet  = '+TempStr,1,30);
            Str(TriangleNormal[1]:6:3,TempStr);
            Str(TriangleNormal[2]:6:3,TempStr1);
            Str(TriangleNormal[3]:6:3,InputStr);
            GRTextXY(' NORMAL =
        '+TempStr+'/'+TempStr1+'/'+InputStr,1,31);
            Str(TriangleVertex[1,1]:6:2,TempStr);
            Str(TriangleVertex[1,2]:6:2,TempStr1);
            Str(TriangleVertex[1,3]:6:2,InputStr);
            GRTextXY(' XYZ PT1=
        '+TempStr+'/'+TempStr1+'/'+InputStr,1,32);
            Str(TriangleVertex[2,1]:6:2,TempStr);
            Str(TriangleVertex[2,2]:6:2,TempStr1);
            Str(TriangleVertex[2,3]:6:2,InputStr);
```

```
            GRTextXY(' XYZ PT2=
'+TempStr+'/'+TempStr1+'/'+InputStr,1,33);
            Str(TriangleVertex[3,1]:6:2,TempStr);
            Str(TriangleVertex[3,2]:6:2,TempStr1);
            Str(TriangleVertex[3,3]:6:2,InputStr);
            GRTextXY(' XYZ PT3=
'+TempStr+'/'+TempStr1+'/'+InputStr,1,34);
         end; { if extended wireframe }

         if DisplayMode='N' then begin
            SelectGraphColor(red);
            SetFillStyle(SolidFill,red);
            FillPoly(4,XRedBox);
            if MonoDisplay then SetColor(Black) else
SelectGraphColor(white);
            InputInt := index;
            Str(InputInt:6,TempStr);
            GRTextXY(' Facet  = '+TempStr,1,30);
            { TriangleNormals should be unit vectors
already. }
            Str(TriangleNormal[1]:6:3,TempStr);
            Str(TriangleNormal[2]:6:3,TempStr1);
            Str(TriangleNormal[3]:6:3,InputStr);
            GRTextXY(' FILE NORM  =
'+TempStr+'/'+TempStr1+'/'+InputStr,1,31);
            for i := 1 to 3 do begin
               Vector3[i]:=TriangleVertex[3,i]-
TriangleVertex[2,i];
               Vector1[i]:=TriangleVertex[1,i]-
TriangleVertex[2,i];
            end;
            { Vector3 cross Vector1 yields correct facet
normal as per SLICE. }
            SliceNormal[1]:= Vector3[2]*Vector1[3]-
Vector3[3]*Vector1[2];
            SliceNormal[2]:= Vector3[3]*Vector1[1]-
Vector3[1]*Vector1[3];
```

```
            SliceNormal[3]:= Vector3[1]*Vector1[2]-
Vector3[2]*Vector1[1];
            barN := SQRT(SQR(SliceNormal[1])+SQR
(SliceNormal[2])
                              +SQR(SliceNormal[3]));
            for i := 1 to 3 do SliceNormal[i] :=
SliceNormal[i]/barN;
            Str(SliceNormal[1]:6:3,TempStr);
            Str(SliceNormal[2]:6:3,TempStr1);
            Str(SliceNormal[3]:6:3,InputStr);
            GRTextXY(' SLICE NORM = '+TempStr+'/'+
TempStr1+'/'+InputStr,1,32);
            NormalsAreOK := true;
            for i:= 1 to 3 do begin
               DeltaNormal[i]:= ABS(TriangleNormal[i]-
SliceNormal[i]);
               if (DeltaNormal[i]>0.001) then
NormalsAreOK := false;
            end;
            if NormalsAreOK then begin
               GRTextXY(' Facet file normal is
correct.',1,34);
            end else begin
               StopDisplayLoop := true;
               GRTextXY(' *** ERROR in file normal
(dashed).',1,34);
               Str(DeltaNormal[1]:6:3,TempStr);
               Str(DeltaNormal[2]:6:3,TempStr1);
               Str(DeltaNormal[3]:6:3,InputStr);
               GRTextXY(' deviation  =
'+ mpStr+'/'+TempStr1+'/'+I putStr,1,33);
               for i:=1 to 3 do begin
                  MidPoint[1,i]:=(TriangleVertex[1,i]+
TriangleVertex[2,i]+
                                    TriangleVertex[3,i])/3.0;
               end;
```

```
                Vertex3DZoom(MidPoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                             saveNormX,saveNormY);
                SelectGraphColor(White);
                OutTextXY(saveNormX-3,saveNormY-3,'+');
                SelectGraphColor(14); {yellow}
                SetLineStyle(DashedLn,$00,NormWidth);
                for i:= 1 to 3 do MidPoint[2,i]:= MidPoint
[1,i]+
                      (TriangleNormal[i]*
(NormLength/100)/Zoom);
                DrawPartLine(Midpoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                             Midpoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
                SetLineStyle(SolidLn,$00,NormWidth);
                for i:= 1 to 3 do MidPoint[2,i]:=
MidPoint[1,i]+
                      (SliceNormal[i]*(NormLength/100)
/Zoom);
                DrawPartLine(Midpoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                             Midpoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
             end;
             SelectGraphColor(CurrentColor);
          end; { if normal check mode  }

          if (DisplayMode <> 'S') or (MonoDisplay) then
begin
             SelectGraphColor(CurrentColor);
             if DisplayMode ='S' then SetLineStyle
(SolidLn,0,NormWidth);
             DrawPartLine(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
                    TriangleVertex[2,1],TriangleVertex
[2,2],TriangleVertex[2,3]);
```

```
            DrawPartLine(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                    TriangleVertex[3,1],TriangleVertex
[3,2],TriangleVertex[3,3]);
            DrawPartLine(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                    TriangleVertex[1,1],TriangleVertex
[1,2],TriangleVertex[1,3]);
         end; { if wireframe }

         if (DisplayMode = 'S') and
(TriangleAxisValue^[index-1]<> 1.2345678) then begin
           { 1.2345678 is a flag indicating the normal
was <= zero }
           Vertex3Dzoom(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
                    GF2D[1,1],GF2D[1,2]);
           Vertex3Dzoom(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                    GF2D[2,1],GF2D[2,2]);
           Vertex3Dzoom(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                    GF2D[3,1],GF2D[3,2]);
           if MonoDisplay then begin
              if CurrentColor=10 then SetFillStyle
(InterleaveFill,1) {FU}
              else if CurrentColor=9 then SetFillStyle
(SlashFill,1) {FD}
              else if CurrentColor=2 then SetFillStyle
(CloseDotFill,1) {U}
              else SetFillStyle(LtSlashFill,1);      {D}
           end else SetFillStyle(SolidFill,CurrentColor);
           FillPoly(3,GF2D);
         end; { if graph is for a surface model }
         if DisplayKey <> 'R' then delay(DisplayDelay);
{    write(gftest,'***');}
```

```
      end;  { is triangleaxisvalue^ in the start to end
range }
{    writeln(gftest,'Process triangle with
INDEX=',index:6,
                  ' TriangleTotal:=',TriangleTotal:6);}
      Inc(Index);
    until (KeyPressed) or
           {(Index>TriangleTotal) or} (stopDisplayLoop)
or (index>=endfacet);
    if KeyPressed then TempChar := upcase(Readkey);
    if TempChar = ESC then exit;
    SelectGraphColor(red);
    if MonoDisplay then SetFillStyle(SolidFill,1)
    else SetFillStyle(SolidFill,red);
    FillPoly(4,RedBox);
    if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
    if Index>=EndFacet then begin
       DrawAxis;
       if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
       GrTextXY(' Drawing Complete: hit any to
exit',1,35);
       repeat delay(100) until KeyPressed;
       TempChar:=upcase(ReadKey);
       DisplayKey:='Q';
    end else begin
       GrTextXY(' Drawing
Paused:Step/Forward/Run?',1,35);
       if (DisplayMode <> 'S') then begin
          repeat
             { FLASHER }
           for i := 1 to 2 do begin
             delay(200);
             if (i = 1) then SelectGraphColor(Black)
             else SelectGraphColor(CurrentColor);
```

```
            if DisplayMode ='S' then
SetLineStyle(SolidLn,0,NormWidth);
            DrawPartLine(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
                  TriangleVertex[2,1],TriangleVertex
[2,2],TriangleVertex[2,3]);
            DrawPartLine(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                  TriangleVertex[3,1],TriangleVertex
[3,2],TriangleVertex[3,3]);
            DrawPartLine(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                  TriangleVertex[1,1],TriangleVertex
[1,2],TriangleVertex[1,3]);
          end;
        until Keypressed;
      end;
      TempChar:= upcase(ReadKey);
      if TempChar=ESC then exit;
      case TempChar of
       {  ' ' : begin
                  repeat delay(100) until KeyPressed;
                  TempChar:=upcase(ReadKey);
                  if TempChar=ESC then exit;
                  DisplayKey:=TempChar;
                end; }
         'Q' : DisplayKey:='Q';
         'R' : DisplayKey:='R';
         'F' : DisplayKey:='F';
         'S' : DisplayKey:='S';
      end;
      if DisplayKey='S' then StopDisplayLoop:=true else
StopDisplayLoop:=false;
    end;
    end; { do while }
    ch := TempChar;
  end;
```

```
procedure PrintRealRange(Str1,Str2 : String;min,max :
Real);
  { Prints Range values on line for parameters }
  begin
    GotoXY(3,WhereY);
    TextColor(LightGray);
    write(Copy(Str1+'              ',1,12),': ');
    TextColor(White);
    if min<MAX_REAL then write(min+0.0003:16:3);
    GotoXY(42,WhereY);
    TextColor(LightGray);
    write(Copy(Str2+'              ',1,12),': ');
    TextColor(White);
    if max>MIN_REAL then write(max+0.0003:16:3);
  end;

procedure PrintTriangleMenu;
  { Print out Triangle Main Menu }
  var
    i  : Word;
  begin
    PrintTriHeader;
    GotoXY(3,5);
    TextColor(LightGray);
    write('Triangle Filename : ');
    TextColor(White);
    write(TriangleFilename);
    GotoXY(47,WhereY);
    TextColor(LightGray);
    write('Triangle Filetype : ');
    TextColor(White);
    case TriangleFiletype of
      ASC_FILE    : write('ASCII');
      BINARY_FILE : write('Binary');
      ARIES_FILE  : write('Aries');
    end;
```

```
      GotoXY(3,WhereY+1);
      TextColor(LightGray);
      write('Total Triangles   : ');
      TextColor(White);
      writeln(TriangleTotal+1);
      for i:= 1 to 3 do begin
        GotoXY(1,WhereY+1);
        PrintRealRange('File '+COOR_CHAR[i]+' Min','File
'+COOR_CHAR[i]+'
Max',TriangleFileMin[i],TriangleFileMax[i]);
      end;
      GotoXY(3,WhereY+3);
      TextColor(LightGray);
      write('Total Selected : ');
      TextColor(White);
      writeln(TriangleSelected);
      PrintRealRange('Selected Min','Selected Max',
        TriangleSelectStart,TriangleSelectEnd);
      for i:= 1 to 3 do begin
        GotoXY(1,WhereY+1);
        PrintRealRange('Sel. '+COOR_CHAR[i]+' Min','Sel.
'+COOR_CHAR[i]+' Max',
          TriangleSelectMin[i],TriangleSelectMax[i]);
      end;
      TextColor(LightGray);
      GotoXY(1,WhereY+2);
      write('   Axis Rotation ( ');
      for i:=1 to 3 do write(COOR_CHAR[i],' ');
      write(') = ');
      TextColor(White);
      for i:= 1 to 3 do write(TriangleAngle[i],' ');
      writeln;
      TextColor(LightGray);
      write('   Windowing by ');
      TextColor(White);
      if TriangleWindowingFile then writeln('File')
      else writeln('Range');
```

```
        TextColor(LightGray);
        write('   Slice Axis is ');
        TextColor(White);
        write(COOR_CHAR[TriangleAxis]);
        GotoXY(49,19);
        TextColor(LightGray);
        write('Display mode=');
        TextColor(White);
        if      DisplayMode='W' then writeln(' Wireframe  ')
        else if DisplayMode='S' then writeln(' Surface    ')
        else if DisplayMode='X' then writeln(' Extended   ')
        else if DisplayMode='N' then writeln('
    NormalCheck');
        GotoXY(21,21);
        TextColor(LightGray);
        write('Horiz Offset=');
        TextColor(White);
        writeln(HoriOffset);
        GotoXY(40,21);
        TextColor(LightGray);
        write('Vertical Offset=');
        TextColor(White);
        writeln(VertOffset);
        GotoXY(62,21);
        TextColor(LightGray);
        write('Zoom = ');
        TextColor(White);
        inputint:=round(zoom*10);
        writeln(InputInt);
        TextColor(LightGray);
        GotoXY(62,20);
        write('Delay= ');
        TextColor(White);
        writeln(DisplayDelay);
        GotoXY(1,23);
        TextColor(LightGray);
```

```
    writeln('Select: New file        Zoom   Window mode
   Select range  Delay');
    writeln('        Toggle display mode   Axis of
viewing   Rotate        Graph');
      write('        Horizontal offset     Vertical
offset   Length        Begin/End');
    CursorOff;
  end;

procedure GetTriangleInfo;
  { Read in Triangle Database }
  var
    FacetCount,index1,index2 :LongInt;
    i,j        : Word;
    CurrentColor: integer;
    realvalue1 : real;
    eyedistance : real;
  begin
    Center('<Scanning Triangle File>',25);
    { Open triangle file }
    Assign(TriangleFile,TriangleFilename);
    ResetFile(TriangleFile);
    TriangleTotal:=0;
    for i:= 1 to 3 do begin
      TriangleFileMin[i]:=MAX_REAL;
      TriangleFileMax[i]:=MIN_REAL;
      TriangleSelectMin[i]:=MAX_REAL;
      TriangleSelectMax[i]:=MIN_REAL;
    end;

    if DisplayMode = 'S' then Init3D;
    repeat
      { Skip over file header }
      FacetCount:=1; { Set in case not Binary file }
      if TriangleFiletype=ASC_FILE then begin
        ReadLine(TriangleFile);
```

```
        if ParseLine(InputLine)<>'SOLID' then
FileError:=TRUE;
      end else begin
        PositionFile(TriangleFile,CurrFilePos+80);
        if TriangleFiletype=BINARY_FILE then begin
          Move(InBuffer[InBufPtr],FacetCount,4);
          PositionFile(TriangleFile,CurrFilePos+4);
        end;
      end;

      { Read in the triangles }
      while (FacetCount>0) and
(TriangleTotal<=Total_Triangles) and
            (Not (FileError or EndOfFile)) do begin
        TrianglePtr^[TriangleTotal]:=CurrFilePos; {
Currently pointing to first triangle }
        if ReadTriangle then begin
          if (TriangleFileType=ARIES_FILE) and
(TriangleNormal[1]=0) and
             (TriangleNormal[2]=0) and
(TriangleNormal[3]=0) then begin
            FacetCount:=0;
            PositionFile(TriangleFile,CurrFilePos+
TriangleAttrib+12);
          end else begin
            if DisplayMode<>'S' then
TriangleAxisValue^[TriangleTotal]:=MAX_REAL
            else
TriangleAxisValue^[TriangleTotal]:=Min_REAL;

          for i := 1 to 3 do begin
            if DisplayMode<>'S' then begin
              if TriangleVertex[i,TriangleAxis]<
TriangleAxisValue^[TriangleTotal]
              then TriangleAxisValue^[TriangleTotal]:=
TriangleVertex[i,TriangleAxis];
            end else begin  { surface mode }
```

```
              eyedistance := ((TriangleVertex[i,1]-
GrOffsetX)*Gr3DMat[0,1])+
                             (TriangleVertex[i,2]-
GrOffsetY)*Gr3DMat[1,1]-
                             (TriangleVertex[i,3]-
GrOffsetZ)*Gr3DMat[2,1];
              if eyedistance>TriangleAxisValue^
[TriangleTotal]
              then TriangleAxisValue^[TriangleTotal]:=
eyedistance;
            end;
            for j:=1 to 3 do begin
              if TriangleVertex[i,j]<TriangleFileMin[j]
then TriangleFileMin[j]:=TriangleVertex[i,j];
              if TriangleVertex[i,j]>TriangleFileMax[j]
then TriangleFileMax[j]:=TriangleVertex[i,j];
              if TriangleVertex[i,j]<
TriangleSelectMin[j] then TriangleSelectMin[j]:=
TriangleVertex[i,j];
              if TriangleVertex[i,j]>
TriangleSelectMax[j] then TriangleSelectMax[j]:=
TriangleVertex[i,j];
            end; { j }
          end;
          Inc(TriangleTotal);
          if TriangleFileType=BINARY_FILE then
Dec(FacetCount);
          { Move to next triangle }
          if TriangleFiletype=ASC_FILE then begin
            ReadLine(TriangleFile);
            if ParseLine(InputLine)<>'ENDLOOP' then
FileError:=TRUE;
            ReadLine(TriangleFIle);
            if ParseLine(InputLine)<>'ENDFACET' then
FileError:=TRUE;
          end else begin
```

```
            Move(InBuffer[InBufPtr+48],TriangleAttrib,2);

            PositionFile(TriangleFile,CurrFilePos+
TriangleAttrib+50);
          end;
           end;  { else from aries file  }
         end;    { if ReadTriangle       }
       end;      { while facetcount>0... }
    until (TriangleTotal>Total_Triangles) or FileError
or EndOfFile;
    if TriangleTotal>Total_Triangles then begin
      Str(TriangleTotal,TempStr);
      Center('*** Warning: Maximum of '+TempStr+'
triangles read in. Ignoring rest. ***',24);
      WaitKey;
    end;
    if Not FileError then begin
      TriangleSelected:=TriangleTotal;
      Dec(TriangleTotal);
      if (DisplayMode = 'S') then begin
        QuickSort(0,TriangleTotal);
        for i:=0 to TriangleTotal do
{           writeln(gftest,' index=',i:6,'
eyedist=',TriangleAxisValue^[i]:8:3);}
        SSorted := true;
      end;
      TriangleSelectStart:=TriangleFileMin
[TriangleAxis];
      TriangleSelectEnd:=TriangleFileMax[TriangleAxis];
     end;

  if Displaymode = 'S' then begin
     { Go through the pointers in their quicksorted
form.  Subsort all
       equal eyedistances by their triangle normal.
After this process the subsorted list of triangles is
ready to be displayed. }
```

```
     TriangleTotal := 0;
     repeat {normalsort}
{       writeln(gftest,'TOP of NORMALSORT, reset
index1=TriTotal');}
       index1:=TriangleTotal;
       eyedistance:=TriangleAxisValue^[TriangleTotal];
{       writeln(gftest,'TriTotal=',TriangleTotal,
                       ' eyedist(index1)=',eyedistance:
8:3,
                       ' TriangleSelected=',
TriangleSelected);
       writeln(gftest,'Eyedist(TriTot+1)=',
TriangleAxisValue^[TriangleTotal+1]:8:3);}
       if (TriangleAxisValue^[TriangleTotal+1]=
eyedistance) and
           (TriangleTotal+1<=TriangleSelected)
       then begin
          repeat
             inc(TriangleTotal);
          until (TriangleAxisValue^[TriangleTotal+1]<>
eyedistance) or
              (TriangleTotal+1>TriangleSelected-1) or
FileError;{ or EndOfFile;}
{          writeln(gftest,'locate index2 normsort.
FileError:=',fileerror,' EndOfFile=',endoffile);}
          index2 := TriangleTotal;
          { Do the sort, which requires computing the
normal for each triangle and storing over where
eyedistance was, then re-sorting from address one to
address two. }
          for i := index1 to index2 do begin
             PositionFile(TriangleFile,TrianglePtr^[i]);
             TempBoolean:=ReadTriangle;

             { this is computing the normal from the eye
view }
```

```
                { we use negative normal to sort back to
front facing }
                TriangleAxisValue^[i]:=
                (   TriangleNormal[1]*Gr3DMat[0,1]
                  + TriangleNormal[2]*Gr3DMat[1,1]
                  - TriangleNormal[3]*Gr3DMat[2,1] );
                if TriangleAxisValue^[i] <= 0
                then TriangleAxisValue^[i] := 1.2345678;
                { this is a flag indicating the facet need
not be drawn }
             end;   { for index1 to index2 }
             quicksort(index1,index2);
             for i:=0 to TriangleSelected-1 do
{              writeln(gftest,' index=',i:6,' normal
=',TriangleAxisValue^[i]:8:3);}
          end;
          inc(TriangleTotal);
        until (TriangleTotal>TriangleSelected-1) or
FileError;{ or EndOfFile; }
{       writeln(gftest,'END of normal sort.
FileError:=',fileerror,' EndOfFile=',endoffile);}
        dec(TriangleTotal);
    end; {normal sort for display mode of surface }
    GrOffsetX:=TriangleFileMin[1]+
((TriangleFileMax[1]-TriangleFileMin[1])/2);
    GrOffsetY:=TriangleFileMin[2]+
((TriangleFileMax[2]-TriangleFileMin[2])/2);
    GrOffsetZ:=TriangleFileMin[3]+
((TriangleFileMax[3]-TriangleFileMin[3])/2);
end;  {procedure GetTriangleInfo}

procedure DoTriangle;
  { User selected Triangle option of Main Menu }
  var
    ch        : Char;
    TempReal1 : Real;
    i,j,k     : Word;
```

```
  begin
    ch := ' ';
    FileError:=FALSE;
    repeat
      if TriangleFilename = '' then ch := 'N'
      else begin
        ResetFile(TriangleFile);
        PrintTriangleMenu;
        if ch=' ' then repeat
          ch:= UpCase(ReadKey);
        until (ch = ESC) or (Pos(ch,'NGTWSRADZHVBEL') >
0);
        TextColor(White);
      end;
      LinePosValid:=FALSE;
      case ch of
        'N' : begin
                CursorOn;
                if GetFilename(TriangleFilename,1) then
begin
                  CursorOff;
                  GotoXY(1,24);
                  write('   File "',TriangleFilename,'"
selected. File type (A)SCII or (B)inary?');
                  repeat
                    TriangleFiletype:=
Pos(UpCase(ReadKey),'ABR');
                  until TriangleFiletype>0;
                  TriangleAxis:=3; { Assume Z }
                  DX := 0;
                  DY := 0;
                  DisplayDelay:=0;
                  Zoom:=1;
                  HoriOffset:=0;
                  VertOffset:=0;
                  NormLength:=500;
                  BeginFacet:=1;
```

```
                    DisplayMode:='W';

                    GetTriangleInfo;

                    EndFacet:= TriangleSelected+1; { Wayno
3.26 }
                    WSorted:=true;
                    SSorted:=false;
                    LinePosValid:=TRUE;
                    TriangleWindowingFile:=TRUE;
                    for i:= 1 to 3 do TriangleAngle[i]:=0;
                    Xpixels := GetMaxX;
                    Ypixels := GetMaxY;
                  end;
                  ch:=' ';
                end;
          'W' : begin
                  if DisplayMode <> 'S'
                  then TriangleWindowingFile:=Not
TriangleWindowingFile;
                  ch:=' ';
                end;
          'T' : begin
                   if DisplayMode='W' then begin
                                                DisplayMode
:='X';
                                                end
                  else if DisplayMode='X' then begin
                                                 DisplayMode
:='N';
                                                 BeginFacet
:=1;
                                                 EndFacet
:=TriangleSelected+1;
                                                    end
                  else if DisplayMode='N' then begin
                                              DisplayMode:='S';
```

```
                                        BeginFacet:= 1;
                                        EndFacet:=
TriangleSelected+1;
                                        end
                  else if DisplayMode='S' then begin
                    DisplayMode:='W';
                    if WSorted = false then begin
                       close(TriangleFile);
                       GetTriangleInfo;
                       WSorted := true;
                    end;
                  end;
                  ch:=' ';
                end;
          'S' : begin
                if DisplayMode <> 'S' then begin
                  GotoXY(1,23);
                  ClrEol;
                  GotoXY(1,24);
                  ClrEol;
                  GotoXY(1,25);
                  ClrEol;
                  GotoXY(1,24);
                  write('   Select Range Start:');
                  CursorOn;
                  readln(TempStr);
                  CursorOff;
                  ToUpper(TempStr);
                  if TempStr='RESET' then begin
                    TriangleSelectStart:=
TriangleFileMin[TriangleAxis];
                    TriangleSelectEnd:=
TriangleFileMax[TriangleAxis];
                  end else begin
                    TempReal := RealVal(TempStr,rc);
                  {   Val(TempStr,TempReal,rc);      }
                    if rc=0 then begin
```

```
                    GotoXY(1,24);
                    ClrEol;
                    write('   Select Range End:');
                    CursorOn;
                    readln(TempStr);
                    CursorOff;
                    TempReal1 := RealVal(TempStr,rc);
                {    Val(TempStr,TempReal1,rc); }
                    if rc=0 then begin
                      if TempReal<TempReal1 then begin
                        TriangleSelectStart:=TempReal;
                        TriangleSelectEnd:=TempReal1;
                      end else begin
                        TriangleSelectStart:=TempReal1;
                        TriangleSelectEnd:=TempReal;
                      end;
                    end;
                  end;
                  for i:=1 to 3 do begin
                    TriangleSelectMin[i]:=MAX_REAL;
                    TriangleSelectMax[i]:=MIN_REAL;
                  end;
                  TriangleSelected:=0;
                  for i:=0 to TriangleTotal do begin
                    if (TriangleAxisValue^[i]>=
TriangleSelectStart) and
                       (TriangleAxisValue^[i]<=
TriangleSelectEnd) then begin
                      PositionFile(TriangleFile,
TrianglePtr^[i]);
                      TempBoolean:=ReadTriangle;
                      TempBoolean:=FALSE;
                      for j:=1 to 3 do
                        for k:= 1 to 3 do begin
                          if TriangleVertex[j,k]<
TriangleSelectMin[k] then TriangleSelectMin[k]:=
TriangleVertex[j,k];
```

```
                              if TriangleVertex[j,k]>
        TriangleSelectMax[k] then TriangleSelectMax[k]:=
        TriangleVertex[j,k];
                            end;
                            Inc(TriangleSelected);
                          end;
                        end;
                      end;
                      end; { displaymode <> 'S' }
                      ch:=' ';
                    end;
            'R' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter new axis rotation in ');
                    for i:=1 to 3 do write(COOR_CHAR[i],'
        ');
                    write(': ');
                    CursorOn;
                    readln(TempStr);
                    CursorOff;
                    GotoXY(1,25);
                    ClrEol;
                    for i:=1 to 3 do begin
                      Val(ParseLine(TempStr),TempReal,rc);
                      if rc=0 then begin
                        TempReal:=Round(TempReal);
                        if (TempReal>=-360) and
        (TempReal<=360) then TriangleAngle[i]:=Round(TempReal);
                      end;
                    end;
                    SSorted := false;
```

```
                    ch:=' ';
                  end;
          'A' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter new axis  ( X, Y, or Z)
');
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    repeat
                      TempByte:=Pos(UpCase(ReadKey),'XYZ');
                    until TempByte>0;
                    if TempByte<>TriangleAxis then begin
                      TriangleAxis:=TempByte;
                      DX:=0;
                      DY:=0;
                      if TriangleAxis=1 then DY:= -90;
{slice on X}
                      if TriangleAxis=2 then DX:= 90;
{slice on Y}
                      for i:= 1 to 3 do TriangleAngle[i]:=
0;
                      SSorted := false;
                    end;
                    ch:=' ';
                  end;
          'D' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
```

```
                write('   Enter delay time (ms) between
triangles: ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >= 0) and
(InputInt<3000)
                then DisplayDelay:=InputInt;
                ch:=' ';
              end;
        'Z' : begin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
                ClrEol;
                GotoXY(1,24);
                write('   Enter Zoom factor (1 to 100,
where no zoom=10): ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >= 1) and
(InputInt<=100)
                then Zoom:=InputInt/10;
                ch:=' ';
              end;
        'V' :  egin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
                ClrEol;
```

```
                GotoXY(1,24);
                write('   Enter Vertical Offset (-10 to
10 screen size): ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >= -1000)
and (InputInt<1000)
                then VertOffset:=InputInt;
                ch:=' ';
              end;
        'H' : begin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
                ClrEol;
                GotoXY(1,24);
                write('   Enter Horizontal Offset (-10
to 10 screen size): ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >= -1000)
and (InputInt<1000)
                then HoriOffset:=InputInt;
                ch:=' ';
              end;
        'L' : begin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
```

```
                    ClrEol;
                    GotoXY(1,24);
                    write('   Length =',normlength,' Enter
new normal factor: ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= 0) and
(InputInt<30000)
                    then NormLength:=InputInt;
                    ch:=' ';
                  end;
          'B' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Begin =',BeginFacet,' Enter
new beginning facet: ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= 1) and
(InputInt<EndFacet)
                    then BeginFacet:=InputInt;
                    ch:=' ';
                  end;
          'E' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
```

```
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   End =',EndFacet,' Enter new
ending facet: ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >
BeginFacet) and (InputInt<=TriangleSelected+1)
                    then EndFacet:=InputInt;
                    ch:=' ';
                  end;
          'G' : begin
                   if (DisplayMode='S') and (not SSorted)
then begin
                      WSorted:=false;
                      close(TriangleFile);
                      GetTriangleInfo;
                   end;
                   { note that if DisplayMode<>'S' we sorted
up at the toggle
                     so that the display on the terminal
screen is correct   }
                   GraphTriangles(ch);
                   ch := Tempchar;
                   TempChar:= 'F';
                   if (Pos(ch,'NGTWSRADZHVBEL') = 0) and
(ch<>ESC) then ch:= ' ';
                   RestoreCrtMode;
                  end;
        end;
      until FileError OR (ch = ESC) OR
(TriangleFilename='');
      if FileError then begin
        if LinePosValid then ASCErrorLn(TriangleFilename)
```

```
      else ASCError(TriangleFilename);
      TriangleFilename:='';
    end;
  end;


{
-------------------------------------------------------
  |                        Start of Main Slice Routines
                  |
  -------------------------------------------------------
}

function FindZLayer(value : Word) : Word;
  { Binary search for Z Value }
  var
    low,mid,high,index  :  Word;
  begin
    low:=0;
    high:=SliceLastLayer;
    index:=MAX_WORD;
    if (value>=SliceLayerValue^[low]) AND
(value<=SliceLayerValue^[high]) then
    while (low<=high) AND (index=MAX_WORD) do begin
      mid:=(low+high) div 2;
      if value<SliceLayerValue^[mid] then high:=mid-1
      else if value>SliceLayerValue^[mid] then
low:=mid+1
           else index:=mid;
    end;
    if index=MAX_WORD then FindZLayer:=high
    else FindZLayer:=index;
  end;


function BlockType(var line : String) : Word;
```

```
{ Returns Block Type of string in Word bits
            B       H       F
           ---     ---     ---
      L     0       1       2      (Bits Set)
      FU    3       4       5
      FD    6       7       8
      NFU   9      10      11
      NFD  12      13      14

      L(level)  = 15
      Not Found = None
}
var
  bitword : Word;
  i       : Word;
begin
  if Copy(line,1,2)='L ' then bitword := L_SECTION
  else begin
    i:=0;
    while (i<MAX_SECTIONS) AND
(Copy(line,1,Length(SliceBlocks[i]))<>SliceBlocks[i]) do
Inc(i);
    if (i=MAX_SECTIONS) OR
(Length(line)=Length(SliceBlocks[i])) then
      bitword := $0000
    else
      case (line[Length(SliceBlocks[i])+1]) of
        'B' : bitword:=1 shl (i*3);
        'H' : bitword:=1 shl (i*3+1);
        'F' : bitword:=1 shl (i*3+2);
        else  bitword:=$0000;
      end;
  end;
  BlockType:=bitword;
end;
```

```
function BitOn(num : Word; bit : Byte) : Boolean;
  { Tells if a bit is on in number }
  begin
    if ((1 shl bit)and num)>0 then BitOn:=TRUE
    else BitOn:=FALSE;
  end;

procedure PrintWordRange(Str1,Str2 : String;min,max :
Word;control : LongInt);
  { Prints Range values on line for parameters }
  begin
    GotoXY(3,WhereY);
    TextColor(LightGray);
    write(Copy(Str1+'              ',1,12),': ');
    TextColor(White);
    if control < 0 then write('Undetermined')
    else write(min);
    GotoXY(53,WhereY);
    TextColor(LightGray);
    write(Copy(Str2+'              ',1,12),': ');
    TextColor(White);
    if control < 0 then write('Undetermined')
    else write(max);
  end;

procedure PrintBlockTable(section : Word);
  { Prints the Block table for menu from section value
at current
     location }
  var
    XPos,YPos   : Byte;
    i,j         : Word;
  begin
    XPos:=WhereX;
    YPos:=WhereY;
    for i:= 0 to 4 do begin
      GotoXY(XPos,YPos+i);
```

```
        TextColor(LightGray);
        write(SliceBlocks[i]);
        GotoXY(XPos+4,WhereY);
        for j:= 0 to 2 do begin
          if BitOn(section,i*3+j) then TextColor(White)
          else if Not MonoDisplay then TextColor(Blue)
               else TextColor(Black);
          if ((i <> 0) or (j <> 2)) then case j of
            0 : write(' B');
            1 : write(' H');
            2 : write(' F');
          end;
        end;
      end;
      TextColor(White);
    end;

procedure PrintSliceMenu;
  { Prints Slice Menu Screen with all information }
  begin
    PrintSliceHeader;
    GotoXY(3,5);
    TextColor(LightGray);
    write('Slice Filename : ');
    TextColor(White);
    write(SliceFilename);
    GotoXY(3,6);
    PrintWordRange('Layer Start','Layer
End',SliceLayerValue^[0],SliceLayerValue^[SliceLastLayer
],1);
    GotoXY(3,7);
    PrintWordRange('File X Min','File X
Max',SliceFileXMin,SliceFileXMax,1);
    GotoXY(3,8);
    PrintWordRange('File Y Min','File Y
Max',SliceFileYMin,SliceFileYMax,1);
    GotoXY(3,10);
```

```
    TextColor(LightGray);
    write('Selected Layer : ');
    TextColor(White);
    if SliceLayer=MAX_WORD then write('None')
    else begin
      write(SliceLayerValue^[SliceLayer]);
      GotoXY(3,11);
      PrintWordRange('Layer X Min','Layer X
Max',SliceLayerXMin^[SliceLayer],SliceLayerXMax^[SliceLa
yer],SliceLayer);
      GotoXY(3,12);
      PrintWordRange('Layer Y Min','Layer Y
Max',SliceLayerYMin^[SliceLayer],SliceLayerYMax^[SliceLa
yer],SliceLayer);
    end;
    GotoXY(3,14);
    TextColor(LightGray);
    write('Windowing by ');
    TextColor(White);
    if SliceWindowingFile then write('File')
    else write('Layer');
    TextColor(LightGray);
    GotoXY(3,16);
    write('Layer Blocks:');
    GotoXY(53,16);
    write('Display Blocks:');
    if SliceLayer<MAX_WORD then begin
      GotoXY(5,17);
      PrintBlockTable(SliceLayerBlocks^[SliceLayer]);
      GotoXY(54,17);
      PrintBlockTable(SliceOutputBlocks);
    end;
    GotoXY(1,23);
    TextColor(White);
    writeln('   Select:  N  new file          L  layer
 select   G  graph  ');
```

```
    write  ('              W  windowing mode    B  block
select              ');
  end;

  function ParseSegment(line : String;var x1,y1,x2,y2 :
Word) : Boolean;
    { Parses out a line that should contain 4 segment
coords. }
    var
      try           : Integer;
      num           : Word;
      good          : Boolean;
      code          : Word;
      oldlen        : Char;
      j             : Word;
    begin
      good:=TRUE;
      try:=1;
      repeat
        if line[0]=#0 then good:=FALSE
        else begin
          if line[1]=' ' then line[1]:='0';
          j:=Pos(' ',line);
          if j=0 then j:=Length(line)+1;
          oldlen := line[0];
          line[0] := Char (j-1);
          Val(line,num,code);
          line[0] := oldlen;
          if code>0 then good:=FALSE
          else case (try) of
            1 : x1:=num;
            2 : y1:=num;
            3 : x2:=num;
            4 : y2:=num;
          end;
          if j>=Length(line) then line:=''
          else begin
```

```
            line[0] := Char (Length(line)-j);
            Move(line[j+1],line[1],Integer (line[0]));
          end;
        end;
        Inc(try);
      until (Not good) OR (try>4);
      ParseSegment := good;
    end;

  procedure GetSliceInfo;
    { Reads a Slice File and sets up Range and Layer
information }
    var
      sectype                : Word;
      x1,y1,x2,y2            : Word;
    begin
      SliceFileXMin  := MAX_WORD;
      SliceFileXMax  := 0;
      SliceFileYMin  := MAX_WORD;
      SliceFileYMax  := 0;
      SliceLayer     := MAX_WORD;
      SliceLastLayer := -1;
      Assign(SliceFile,SliceFilename);
      Reset(SliceFile,1);
      ResetFile(SliceFile);
      ReadLine(SliceFile);
      repeat
        if (Not EndOfFile) AND (InputLine[1]<>'!') then
begin
          sectype := BlockType(InputLine);
          if sectype = 0 then
            if (Not ParseSegment(InputLine,x1,y1,x2,y2))
or (SliceLastLayer<0) then FileError:=TRUE
            else begin
              { A Segment }
              SetMinMax(x1,SliceLayerXMin^
[SliceLastLayer],SliceLayerXMax^[SliceLastLayer]);
```

```
                SetMinMax(x2,SliceLayerXMin^
[SliceLastLayer],SliceLayerXMax^[SliceLastLayer]);
                SetMinMax(y1,SliceLayerYMin^
[SliceLastLayer],SliceLayerYMax^[SliceLastLayer]);
                SetMinMax(y2,SliceLayerYMin^
[SliceLastLayer],SliceLayerYMax^[SliceLastLayer]);
              end
            else
              { A Block line }
              if sectype=L_SECTION then begin
                if SliceLastLayer>=0 then begin
                  if SliceLayerXMin^[SliceLastLayer]<
SliceFileXMin then SliceFileXMin:=SliceLayerXMin^
[SliceLastLayer];
                  if SliceLayerXMax^[SliceLastLayer]>
SliceFileXMax then SliceFileXMax:=SliceLayerXMax^
[SliceLastLayer];
                  if SliceLayerYMin^[SliceLastLayer]<
SliceFileYMin then SliceFileYMin:=SliceLayerYMin^
[SliceLastLayer];
                  if SliceLayerYMax^[SliceLastLayer]>
SliceFileYMax then SliceFileYMax:=SliceLayerYMax^
[SliceLastLayer];
                end;
                Inc(SliceLastLayer);
                if SliceLastLayer>Total_Layers then begin
                  Str(SliceLastLayer,TempStr);
                  Center('Warning - Maximum number of
'+TempStr+' Slice layers read',24);
                  WaitKey;
                end else begin
                  Val(Copy(InputLine,3,255),TempWord,rc);
                  if rc<>0 then FileError:=TRUE
                  else begin
                    SliceLayerValue^[SliceLastLayer]:=
TempWord;
```

```
                    SliceLayerBlocks^[SliceLastLayer]:=
$0000;
                    SliceLayerPtr^[SliceLastLayer]:=
CurrFilePos;
                    SliceLayerXMin^[SliceLastLayer]:=
MAX_WORD;
                    SliceLayerXMax^[SliceLastLayer]:=0;
                    SliceLayerYMin^[SliceLastLayer]:=
MAX_WORD;
                    SliceLayerYMax^[SliceLastLayer]:=0;
                  end;
                end;
              end else SliceLayerBlocks^[SliceLastLayer]:=
SliceLayerBlocks^[SliceLastLayer] or sectype;
          end;
          ReadLine(SliceFile);
        until FileError OR EndOfFile OR (SliceLastLayer>
Total_Layers);
        if Not FileError then begin
          if SliceLastLayer>Total_Layers then
SliceLastLayer:=Total_Layers;
          if SliceLayerXMin^[SliceLastLayer]<SliceFileXMin
then SliceFileXMin:=SliceLayerXMin^[SliceLastLayer];
          if SliceLayerXMax^[SliceLastLayer]>SliceFileXMax
then SliceFileXMax:=SliceLayerXMax^[SliceLastLayer];
          if SliceLayerYMin^[SliceLastLayer]<SliceFileYMin
then SliceFileYMin:=SliceLayerYMin^[SliceLastLayer];
          if SliceLayerYMax^[SliceLastLayer]>SliceFileYMax
then SliceFileYMax:=SliceLayerYMax^[SliceLastLayer];
        end;
      end:

procedure GraphSliceLayer;
  var
    sectype,currentsection : Word;
    x1,y1,x2,y2            : Word;
  begin
```

```
    if SliceWindowingFile then
      SetupGraphicW(SliceFileXMin,SliceFileXMax,
SliceFileYMin,SliceFileYMax)
    else
      SetupGraphicW(SliceLayerXMin^[SliceLayer],
SliceLayerXMax^[SliceLayer],
                    SliceLayerYMin^[SliceLayer],
SliceLayerYMax^[SliceLayer]);
    OutTextLn('Filename: '+SliceFIlename);
    Str(SliceLayerValue^[SliceLayer],TempStr);
    OutTextLn('Layer    : '+TempStr);
    GrTextXY('press Q to quit display-',43,1);
    GrTextXY('or <esc> for main menu',43,2);


    PositionFile(SliceFile,SliceLayerPtr^[SliceLayer]);
    repeat
      ReadLine(SliceFile);
      if (Not EndOfFile) AND (InputLine[1]<>'!') then
begin
        sectype := BlockType(InputLine);
        if sectype=0 then begin
          { A Segment }
          if (currentsection>0) AND
ParseSegment(InputLine,x1,y1,x2,y2) then
DrawLine(x1,y1,x2,y2);
        end else begin
          { A Block line }
          currentsection:=sectype AND SliceOutputBlocks;
          if currentsection>0 then
            if currentsection=LB_SECTION then
SelectGraphColor(Blue)
            else if currentsection=LH_SECTION then
SelectGraphColor(LightBlue)
            else if (currentsection and NFD_SECTIONS)>0
then SelectGraphColor(Red)
```

```
            else if (currentsection and NFU_SECTIONS)>0
then SelectGraphColor(Green)
            else if (currentsection and FD_SECTIONS)>0
then SelectGraphColor(LightRed)
            else if (currentsection and FU_SECTIONS)>0
then SelectGraphColor(LightGreen)
            else SelectGraphColor(White);
        end;
      end;
    until (sectype=L_SECTION) OR EndOfFile;
  end;


procedure DoSlice;
  { User selected Slice option of Main Menu }
  var
    ch  : Char;
    i,j : Word;
    index : Integer;
  begin
    ch := ' ';
    repeat
      if SliceFilename = '' then ch := 'N'
      else begin
        ResetFile(SliceFile);
        PrintSliceMenu;
        if ch=' ' then repeat
          ch:= UpCase(ReadKey);
        until (ch = ESC) or (Pos(ch,'NLBWG') > 0);
        TextColor(White);
      end;
      LinePosValid:=FALSE;
      case ch of
        'N' : begin
                if GetFilename(SliceFilename,2) then
begin
                  Center('<Scanning Slice File>',25);
```

```
                    GetSliceInfo;
                    LinePosValid:=TRUE;
                    SliceWindowingFile:=TRUE;
                    SliceLayer:=0;
                    SliceOutputBlocks:=SliceLayerBlocks^
[SliceLayer];
                  end;
                  ch:=' ';
                end;
          'L' : begin
                  index:=0;
                  TempInt:=1;
                  repeat
                    if TempInt<>index then begin
                      ClrScr;
                      Center('Slice Layer Select',1);
                      for j:=0 to 15 do if j+index<=
SliceLastLayer then begin
                        GotoXY(1+(j mod 4)*20,3+(j div
4)*5);
                        write(SliceLayerValue^[j+index]:
5,' ');
                        PrintBlockTable(SliceLayerBlocks^
[j+index]);
                      end;
                      Center('PgUp or PgDn to move
backward or forward. ENTER to select Level',24);
                      TempInt:=index;
                    end;
                    ch:=ReadKey;
                    if ch=#0 then begin
                      ch:=ReadKey;
                      if (ch=#81) and
(index+16<=SliceLastLayer) then Inc(index,16);
                      if (ch=#73) and (index-16>=0) then
Dec(index,16);
                    end;
```

```
                    until ch=CR;
                    GotoXY(1,24);
                    ClrEol;
                    write('Enter Layer Number: ');
                    readln(TempStr);
                    if TempStr<>'' then begin
                      Val(TempStr,iw,rc);
                      jw:=FindZLayer(iw);
                      if (rc<>0) OR (jw=MAX_WORD) then begin
                        Center('Layer "'+TempStr+'" Not
Found',24);
                        WaitKey;
                      end else begin
                        SliceLayer:=jw;
                        SliceOutputBlocks:=SliceLayerBlocks^
[SliceLayer];
                      end;
                    end;
                    ch:=' ';
                  end;
          'B' : begin
                    GotoXY(1,24);
                    ClrEol;
                    write('Enter Block Type to Toggle: ');
                    readln(TempStr);
                    ToUpper(TempStr);
                    SliceOutputBlocks:=SliceOutputBlocks xor
BlockType(TempStr);
                    ch:=' ';
                  end;
          'W' : begin
                    SliceWindowingFile := not
SliceWindowingFile;
                    ch:=' ';
                  end;
          'G' : if SliceLayer<MAX_WORD then begin
                    repeat
```

```
                    GraphSliceLayer;
                    ch:=UpCase(ReadKey);
                  until ch<>'G';
                  RestoreCRTMode;
                end else ch:=' ';
          ELSE if ch<>ESC then ch:=' ';
        end;
      until (ch = ESC) OR (SliceFilename='');
      if FileError then begin
        if LinePosValid then ASCErrorLn(TriangleFilename)
        else ASCError(TriangleFilename);
        TriangleFilename:='';
      end;
    end;


{-------------------------------------------------------
  |                          Start of Main Program
                  |
  -------------------------------------------------------
}

  begin
    { Setup Exit routine }
    ExitSave:=ExitProc;
    ExitProc:=@MainExit;

    { Determine graphics screen to use }
    GraphDriver := Detect;
    SetGraph;
    RestoreCrtMode;
{    assign(gftest,'gftest.txt');
    rewrite(gftest); }

    { Use this heap location as top of heap for rest of
program }
```

```
      Mark(HeapTop);

      { Allocate Memory for arrays }
      ReAllocate('S',16382);
      if Total_Triangles<5000 then ReAllocate('T',5000);

      repeat
        PrintHeader;
        Center('Graphic Card:
'+GraphDriverName(GraphDriver),WhereY+2);
        Str(GraphMode,TempStr1);
        TempStr:='Graphics Mode:'+TempStr1+' (';
        Str(SizeX,TempStr1);
        TempStr:=TempStr+TempStr1+',';
        Str(SizeY,TempStr1);
        TempStr:=TempStr+TempStr1+')';
        Center(TempStr,WhereY+1);
        GotoXY(1,25);
        write('Maximum Triangles: ',Total_Triangles+1);
        GotoXY(57,25);
        write('Maximum Layers: ',Total_Layers+1);


        GotoXY(22,10);
        write('1. Graph Triangles ( .STL file )');
        GotoXY(22,WhereY+2);
        write('2. Graph Slice Layers ( .SLI file )');
        GotoXY(22,WhereY+2);
        write('3. Change Triangles / Layer Allocation');
        GotoXY(22,WhereY+2);
        write('Q. Quit program');

        GotoXY(28,WhereY+3);
        write('Enter Selection: ');
        repeat
          ch := UpCase(ReadKey);
        until Pos(ch,'123Q')>0;
```

```
      Case ch of
        '1' : if Total_Triangles>=0 then DoTriangle;
        '2' : if Total_Layers>=0 then DoSlice;
        '3' : begin
                ClrScr;
                GotoXY(1,24);
                write('Enter number of Slice Layers
desired: ');
                readln(TempStr);
                if TempStr<>'' then begin
                  Val(TempStr,TempReal,rc);
                  TempLong:=Round(TempReal);
                  if (rc=0) and (TempLong>=0) then
ReAllocate('S',TempLong);
                end;
              end;
      end;
    until (ch = 'Q');
    TextBackground(Black);
    TextColor(White);
{    close(gftest); }
    CursorOn;
    ClrScr;
  end.
```

```
{=========================================================
      3D Systems Stereolithography System Software

=========================================================
                        BEAM.PAS
      SLA-1 Beam Profile and Calibration Software
 -------------------------------------------------

   Recent History:

   11/23/87  Ver 2.30  R&D PROFILE3 adapted to Menu
                       System from profile6.bas
   12/4/87   Ver 1.0+=
   12/06/87  Ver 2.35  updated to operate like R&D
                       PROFILE6
   1/18/88  Ver 2.50   Beta Release
    2/05/88  Ver 2.60  sensor saturation message is
                       followed by delay instead
                       of keyboard pause to let
                       Calibration routine complete
                       maximum ADC value shown on beam
                       profile screen
                       /NUM option shows profile as
                       numbers instead of dots
                       BEAM.PRM given options line; old
                       PRM format supported
    2/17/88  Ver 2.61  /NUM option changed to /TEXT
                       option /TEXT option affects
                       option 5 (Focus Params) too
                       menu toggle between text and
                       graphics format
                       size of dots reduced relative to
                       ADC value
                       spiral loop begins at -5, not +1
    2/22/88  Ver 2.62  calibration patches
                       erasure of column and improved
                       pause countdown
```

```
3/02/88  Ver 2.80  BEAM converted to Turbo Pascal
    Version 4.0    simple structural and operational
                   changes beam read and display
                   separated out
                   single beam profile routine
3/05/88  Ver 2.81  verification of 2 mil hole
                   sensors 1 & 2 not scanned at
                   start-up
                   new calibration algorithm
3/08/88  Ver 2.82  improved calibration routine
                   fast, medium, slow & different
                   calib speeds
                   graphic beam profile uses
                   un-filled circles
3/09/88  Ver 2.83  better timeouts for non-existent
                   hardware
                   all Z-Stage code removed
                   (obsoleted)
                   added copyright notice
3/14/88  Ver 2.84  calibration timestamp fixed
                   FindBeam reports found beams
                   properly
3/25/88  Ver 2.841 Modify XSensorPos and YSensorPos
                   to a real array
                   to prevent overflow which might
                   result from its
                   previous type used (word array)
4/05/88  Ver 2.85  Modify Calibration such that
                   UpdateCalibFileCol
                   occurs after GoodCol
                   UpdateCalibFileRow uses -1 for
                   Xsp[1]....Ysp[2]
                   so that sensors track drift
4/08/88  Ver 2.86  added variable time delay (T then
                   0-9) between
                   position check readings
                   added X/Y position select for
```

```
                                position check
4/08/88   Ver 2.87   Position Check lands in middle
                     before seeking new
                     profile sensor position
4/09/88   Ver 3.00   production release
4/11/88   Ver 3.01   adding timeout to profiler search
                     in calibration
                     display calibration position on
                     screen
                     UpdateCalibFileCol moved to
                     better point in
                     calibration routine
                     Slow method of position check
                     (toggle select)
4/13/88   Ver 3.02   Slow position check method chosen
                     as default
                     position check method and display
                     type hidden
4/15/88   Ver 3.03   added Row and Col delay to Power
                     function
                     made a true moving average in
                     Power function
4/18/88   Ver 3.04   LongInt variables in slow
                     position check alg
4/27/88   Ver 3.05   laser beam movement slew rate
                     limiting implemented
5/06/88   Ver 3.06   fixed background noise set for
                     some routines
                     (use findHoles2 and modify
                      beamscan.inc- added in 3.12)
5/02/88   Ver 3.10   laser on check, auto-turn off
5/31/88   Ver 3.11   Add calibration routine menu and
                     related enhancements
6/01/88   Ver 3.12   Find pinholes before displaying
                     main beam menu
6/15/88   Ver 3.13   Correct beam power algorithm,
                     double the
```

```
                        profile resolution, and add
                        averaging of power.
    6/21/88  Ver 3.14   go back to normal resolution
                        (22x22) and reinstate
                        option 4, the power averaging
                        function
    7/06/88  Ver 3.21   Stuartian enhancements
    8/19/88  Ver 3.22   Change E2Pot to 180 - x iff > 0
                         as per hardware change
    9/01/88  Ver 3.221 Change Calib. Operations to take
                        *.GEO or *
    9/01/88  Ver 3.222 Correct FOCUS5 1)runtime err
                        2)default sensor 2  3)size
    9/06/88  Ver 3.223 Calibration fixes. 1.reset middle
                        2. three tries in CalibFind and
                        stepsize = 9
    9/10/88  Ver 3.224 Add exits and final changes to
                        Calibration.
    9/13/88  Ver 3.23   add PROFMOVE and sensor
                        background just off sensor hole
    9/14/88  Ver 3.24   add exit to adjust gain (calibr8)
                        when profile not found
 ------------------------------------------------------
    Future Mods:  (Review)
    Watch temperature near 10C option base 0, 1-13,2-13,
       generate the R#(0)=r#(1)*2-r#(2), etc t#
   Should be able to define Move6 position as new center
   Also check for temp Blanket regulation
        by going to 2 different power locations (see
STS)
        and watching drift settle (typ 10 seconds)
   Spread printout when drifts near zero
   Utilize the 4 mil, etc. holes in calculating beam
power/profile.
------------------------------------------------------}
{$R-,S-,I-,D+,T-,F-,V-,B-,N-,L- }
{$M 16384,0,16384 }
```

```
program Beam;
uses Crt,
     Dos,
     Graph,
     Drivers;
type
  str6  = string[6];
  str10 = string[10];
  str16 = string[16];
  str80 = string[80];
  OpModeType = (TextOp,GraphicsOp);
  BeamProfileVarSensorType = record
    Xpos             : LongInt;    { inputs }
    Ypos             : LongInt;
    ScanStep         : Integer;
    BkgndThreshold   : Real;
    InitialDelay     : Integer;
    RowToRowDelay    : Integer;
    ColToColDelay    : Integer;
    XYweight         : Real;       { outputs }
    Xweight          : Real;
    Yweight          : Real;
    TotalWeight      : Real;
    HiPoints         : Word;
    SaturationCount  : Integer;
    MaxSensorReading: Integer;
    BeamMap          : array[0..44,0..44] of Real;
    RowWeight        : array[0..44] of Real;
    ColWeight        : array[0..44] of Real;
    AveBackground    : real;
  end;
  BeamProfileVarType = array[1..3] of
BeamProfileVarSensorType;
  FindBeamVarSensorType = record
    Xval             : LongInt;    { inputs & updated }
    Yval             : LongInt;
    SpiralDelay      : Word;       { inputs }
```

```
    SpiralSideMax   : Integer;
    SpiralTries     : Integer;
    ScanStep        : Integer;
    BkgndThreshold  : Integer;
    NumCenterPasses : Integer;
    Xavg            : Real;      { outputs }
    Yavg            : Real;
  end;
  FindBeamVarType = array[1..3] of
FindBeamVarSensorType;
  FindMethodType = (none,NormalFind,Verify2MilHole);
  PCM_Mode = (no_PCM,GainDrift,StraightXY);
const
  VersionId = '3.24';
  SoundOn   = false;
  ParamFileName          = 'BEAM.PRM';
  ErrorFileName          = 'C:\SYSTEM.ERR';
  HomeDirectory          = 'C:\3DSYS\';
  PowerHistoryFileName   = 'C:\HISTORY\POWER.MON';
  ProfileHistoryFileName = 'C:\HISTORY\PROFILE.MON';
  Blank  = '
                                ';
  ColMax = 22;
  RowMax = 22;
  PowerMvAvg = 100;
   CtrlC = ^C;
  Bs    = ^H;
  Cr    = ^M;
   CalibSelect: array[1..3] of str6 = ('  1   ','1 or
      2','1 - 3 ');
  LaserAddr    = $300;
  PowerPort    = $304;
  _INIT_IN     = $10;
  NumSensors   = 3;
  DiscreteBeam = 5;
  Timeout      = 60000;
  lamda        = 5.9;
```

```
var
  FirstVar: Byte;  { must be first variable defined }
  { beam arrays }
  Resistance,Temperature: array[0..100] of Real;
  { sensor arrays }
  SensorChannel: array[1..12] of Word;
  XSensorPos,YSensorPos,SensorBkgnd : array[1..3] of
Real;
  Power,Bk: array[1..3] of Real;
  Xsp,Ysp: array[1..3] of Real;
  SensorCalib: array[1..3,1..4] of Real;
  Hist2D: array[1..4,1..2] of Integer;
  LastField: array[1..4] of Integer;
  { miscellaneous }
  FNTimeStamp : str10;
  CalibFileName : str80;
  key: Char;
  i  : Integer;
  AverageBeamPowerArray : array[1..10] of real;
  PrintDev,CalOutFile,ErrorFile,ParamFile: Text;
  ZTableOff,LaserOff,ShowStuff,DemoMode: Boolean;

PROFMOVE,BORDERSOK,GrInstalled,Debug,AbortFlag,PrintOut,
Display,Calibrate: Boolean;
  BeamFound: Boolean;
  OutFileName,SingleBar,CmdLine: String;
  XRest,YRest,oldX,oldY: Word;
  TempChannel,TempCalPts: Word;
  Sensor,Select,SensorNum,PosChkSpeed: Integer;
  PeakReading : real;
  Xpos,Ypos: LongInt;
  Xaverage,Yaverage,TotWtAvg: Real;
  GraphDriver,GraphMode,Error,code: Integer;
  ViewPort: ViewPortType;
  OpMode: OpModeType;
  OldExitProc: Pointer;
  regs: Registers;
```

```
  BeamProfileVariables: BeamProfileVarType;
  FindBeamVariables: FindBeamVarType;
  Display1Sensor,Focus5Sensor,GoodCount: Integer;
  CurrentDate,CurrentTime: String;
  Sec,Minu,Hour,Day,WeekDay,Mon,Year: Integer;
  CalibDelays: array[1..3,1..3] of LongInt;
  PositionCheckMode: PCM_Mode;
  SlewLimitMaxStep,SlewLimitDelay: Word;
  junkkey : char;
  LastVar: Byte;  { should be last variable defined }
procedure Abort; forward;
{-------------------------------------------------}
{$i beamutil.inc}
{$i beamgr.inc}
{$i beamio.inc}
{$i beamscan.inc}
{  $i beamcali.inc--  this is moved down to where the
CALIBR8 proc was. }
{-------------------------------------------------}
function HardwareLaserOn: Boolean;
const
  PowerPort = $304;
  _LASER_IN = $80;
begin
  if port[PowerPort + 2] and _LASER_IN = 0 then
HardwareLaserOn := true
    else HardwareLaserOn := false;
end;
{
   check for BEAM Program Options
}
procedure CheckForOptions(var CmdLine: String);
var
  i: Integer;
begin
  CmdLine := upper(CmdLine);
  i := pos('/DEBUG',CmdLine);
```

```
  if i>0 then begin
    Debug := true;
    delete(CmdLine,i,6);
  end;
  i := pos('/TEXT',CmdLine);
  if i>0 then begin
    OpMode := TextOp;
    delete(CmdLine,i,5);
  end;
  i := pos('/DEMO',CmdLine);
  if i>0 then begin
    DemoMode := true;
    delete(CmdLine,i,5);
  end;
  i := pos('/S',CmdLine);
  if i>0 then begin   { check for /S Show Stuff flag }
    ShowStuff := true;
    delete(CmdLine,i,2);
  end;
  i := pos('/LOFF',CmdLine);   { check for /LOFF No
Laser flag }
  if i>0 then begin
    LaserOff := true;
    delete(CmdLine,i,5);
  end;
end;
{
  Read BEAM.PRM parameters file
}
procedure ReadBeamParamsFile;
var
  ParamFile: text;
  i,count,code,code1,code2,Sensor,Pinhole: Integer;
  cmd,ParamLine: String;
   procedure GetParamFileLine;
  begin
    if eof(ParamFile) then ParamLine := ''
```

```
    else repeat
      readln(ParamFile,ParamLine);
    until eof(ParamFile) or ((ParamLine <> '') and
(ParamLine[1] <> '!'));
  end;
 begin
  Center('--- Reading Parameters File ---',23);
  {$I-}
  assign(ParamFile,ParamFileName);
  reset(ParamFile);
  if IOresult<>0 then begin
    assign(ParamFile,HomeDirectory+ParamFileName);
    reset(ParamFile);
    if IOresult<>0 then begin
      Center('Can''t find Beam Parameter file.',24);
      WaitKey;
      halt(3);
    end;
  end;
  {$I+}
  GetParamFileLine;
  if copy(ParamLine,1,1) = chr(34) then begin   { handle
quoted options line }
    i := pos(chr(34),copy(ParamLine,2,255));
    if i>0 then ParamLine[0] := chr(i);
    ParamLine := NoQuotes(ParamLine);
    CheckForOptions(ParamLine);
    GetParamFileLine;
  end;
  XRest := WordVal(ParamLine,code);
  YRest := WordVal(copy(ParamLine,code+1,255),code);
  GetParamFileLine;
  SensorNum := WordVal(ParamLine,code);
  for Sensor:=1 to SensorNum do begin
    GetParamFileLine;
    SensorChannel[Sensor] := WordVal(ParamLine,code);
    GetParamFileLine;
```

```
      XSensorPos[Sensor] := WordVal(ParamLine,code);
      YSensorPos[Sensor] :=
WordVal(copy(ParamLine,code+1,255),code);
      for Pinhole:=1 to 4 do begin
        GetParamFileLine;
        SensorCalib[Sensor,Pinhole] :=
RealVal(ParamLine,code);
      end;
    end; {for Sensor}
    GetParamFileLine;
    TempChannel := WordVal(ParamLine,code);    { assume
only one thermometer at first }
    GetParamFileLine;
    TempCalPts := WordVal(ParamLine,code);
    for i:=1 to TempCalPts do begin
      { Resistance[i],Temperature[i] may not be integers }
      GetParamFileLine;
      Resistance [i] := RealVal(ParamLine,code);
      Temperature[i] :=
RealVal(copy(ParamLine,code+1,255),code) + 273.15;
    end;
    for i:=1 to 3 do begin    { read in Fast, Medium, &
Slow Calib Delay values }
      GetParamFileLine;
      CalibDelays[i,1] := LongVal(ParamLine,code1);
      CalibDelays[i,2] :=
trunc(RealVal(copy(ParamLine,code1+1,255),code2)*10+0.5)
;
      CalibDelays[i,3] :=
trunc(RealVal(copy(ParamLine,code2+code1+1,255),code)*10
+0.5);
    end;
    SlewLimitMaxStep := 1000;
    SlewLimitDelay   := 2;
    if not eof(ParamFile) then begin
      GetParamFileLine;
      SlewLimitMaxStep := WordVal(ParamLine,code);
```

```
    if SlewLimitMaxStep < 10 then SlewLimitMaxStep :=
10;
    GetParamFileLine;
    SlewLimitDelay    := WordVal(ParamLine,code);
  end;
  close(ParamFile);
  Resistance [0] := Resistance[1]  * 2 - Resistance [2];
  Temperature[0] := Temperature[1] * 2 - Temperature[2];
end;
{
   ****************************
   * Option 1: Display Profile *
   ****************************
}
procedure Display1;
var
  key: Char;
  v,code: Integer;
  Leave,NewSensor: Boolean;
begin
  Sensor := Display1Sensor;
  Display := true;
  Leave := false;
  repeat
    SetScreen;
    gotoxy(1,25);
    write('(1) DISPLAY PROFILE: Press
',CalibSelect[NumSensors],
      ' to select sensor #, Q to Quit');
    NewSensor := false;
    repeat
      Display := true;
      BeamProfileVariables[Sensor].ScanStep := 2;
      GoodRead(1);
      if TotWtAvg=0 then begin
        Center(' ERROR: Sensor #'+IntStr(Sensor)+
```

```
          ' was not found: Try (6) Move or (7)
Search',24);
        WaitKey;
        exit;
      end;
      if GoodCount > 0 then begin
        Xsp[Sensor] := trunc(Xaverage+0.5);
        Ysp[Sensor] := trunc(Yaverage+0.5);
      end;
      PrintInfo(Sensor);
      if keypressed then begin
        key := ReadKey;
        v := IntVal(key,code);
        if (v > 0) and (v <= NumSensors) then begin
          Sensor := v;
          PROFMOVE := true;

MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
          PROFMOVE := false;
          delay(200);
          NewSensor := true;
        end else Leave := true;
      end;
    until NewSensor or Leave;
  until Leave;
  Display1Sensor := Sensor;
end;
{
   **********************
   * Option 2: History *
   **********************
}
procedure ShowHistory(FileName: String);
var
  HistoryFile: Text;
  IOerr: Integer;
begin
```

```
  assign(HistoryFile,FileName);
  {$I-}
  reset(HistoryFile);
  IOerr := IOresult;
  {$I+}
  case IOerr of
    0: begin
         Exec('C:\COMMAND.COM','/C SCAN ' + FileName);
       end;
    2: begin
         Center(FileName + ' not yet created.',24);
         WaitKey;
       end;
    else begin
      Center('Error ' + IntStr(IOerr) + ' trying to read
' + FileName + '.',24);
      WaitKey;
    end;
  end;
end;
procedure ShowPowers;
begin
  ShowHistory(PowerHistoryFileName);
end;
procedure ShowProfiles;
begin
  ShowHistory(ProfileHistoryFileName);
end;

procedure History2;
begin
  gotoxy(1,24);
  writeln('(2) History of Beam Power & Profile');
  gotoxy(1,15);
  write('Review (1) Beam Power or (2) Profile? ');
  key := ReadKey;
  if key in ['1','2'] then write(key);
```

```
  case IntVal(key,code) of
    1: ShowPowers;
    2: ShowProfiles;
  end;
end;
{
   ****************************
   * Option 3: Position Check *
   ****************************
}
procedure Ranger(ffield: Integer;position: Real;fch:
Char);   { field 1-4 }
var
  ch: Char;
  CurrField,PPos: Integer;
begin
  CurrField := trunc(3*(position+80000.0)/19);   { 3
character positions per .001' }
  PPos := trunc(3*(position+80000.0)-CurrField*19.0);
  ch := fch;
  if LastField[ffield] < 32767 then begin   { always
display first one }
    if CurrField < LastField[ffield] then   { see if in
new field }
      ch := '/'
    else if CurrField > LastField[ffield] then ch :=
'\';
  end;
  Last ield[ffield] := CurrField;
  PPos  = PPos+1+20*(ffield-1);
  gotc  (PPos,20);
  write ch);
  gotoxy(20*ffield,20);
  write('|');
  if PrintOut then begin
```

```
write(PrintDev,spc(PPos),ch,chr(13),spc(20*ffield),'|',c
hr(13));
    if ffield=4 then writeln(PrintDev);
  end;
end;
procedure Position3;
var
  i,step: Integer;
  x,y,dir: LongInt;
  temp: String;
  Xoffs,Yoffs,Xgain,Ygain: Real;
  procedure PrinterOff;
  begin
    if printout then begin
      writeln(PrintDev,TimeStamp);
  { restore 1/6 line spacing, non-condensed }
      write(PrintDev,chr(27),'A',chr(12),chr(18));
      write(PrintDev,chr(12));
      close(PrintDev);
      PrintOut := false;
    end;
  end;

begin
  for i:=1 to 4 do   { set up initial fields for Ranger
}
    LastField[i] := 32767;
  Display := false;
  repeat
    repeat
      gotoxy(1,21);
      if PositionCheckMode = GainDrift then
        writeln('         X Offset           X Gain
       Y Offset            Y Gain')
      else
```

```
        writeln('            X1                    Y1
          X2                  Y2  ');
      if PositionCheckMode = GainDrift then temp :=
'Gain/Drift Mode'
        else temp := 'Straight X/Y Mode';
      Center(temp,22);
      writeln;
      write(Blank);
      for Sensor:=1 to 2 do begin
        gotoxy(20,22+Sensor);
        write('Sensor #',Sensor,Power[Sensor]:5:1,' mW,
Center: ');
        PrintXY(Xsp[Sensor],Ysp[Sensor]);
        write('      ');
      end; {Sensor}
      Center('(3) POSITION CHECK: P-Printout  0-9-Speed
Q-Quit',25);

      for Sensor:=1 to 2 do begin
        Display := false;
        BeamProfileVariables[Sensor].ScanStep := 4;
        PROFMOVE := true;
        MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
        PROFMOVE := false;
        GoodRead(1);
        if GoodCount > 0 then begin
          Xsp[Sensor] := trunc(Xaverage+0.5);   { update
centers }
          Ysp[Sensor] := trunc(Yaverage+0.5);
        end;
      end;  {Sensor}
      Xoffs := (
Xsp[1]+Xsp[2]-XSensorPos[1]-XSensorPos[2] )/(2*3.556);
{mils}
      Yoffs := (
Ysp[1]+Ysp[2]-YSensorPos[1]-YSensorPos[2] )/(2*3.556);
```

```
      Xgain :=
((Xsp[1]-Xsp[2])/(XSensorPos[1]-XSensorPos[2])-1)*10000;
{mils per 10'}
      Ygain :=
((Ysp[1]-Ysp[2])/(YSensorPos[1]-YSensorPos[2])-1)*10000;
      window(1,1,80,20);
      gotoxy(1,20);
      writeln;
      window(1,1,80,25);
      if PositionCheckMode = GainDrift then begin
        Ranger(1,Xoffs,'X');
        Ranger(2,Xgain,'x');
        Ranger(3,Yoffs,'Y');
        Ranger(4,Ygain,'y');
      end else begin
        Ranger(1,Xsp[1],'X');
        Ranger(2,Ysp[1],'Y');
        Ranger(3,Xsp[2],'x');
        Ranger(4,Ysp[2],'y');
      end;
      if not keypressed then delay(PosChkSpeed);
    until keypressed;
    key := upcase(ReadKey);
    case key of
      'P': begin
             gotoxy(1,1);
             Center('Do you want a printout (y/n)
?',25);
             if (upcase(ReadKey) = 'Y') and not PrintOut
then begin
               PrintOut := true;
               assign(PrintDev,'PRN');
               rewrite(PrintDev);
               writeln(PrintDev,'X is offset, x is
length change of 10',
                 '" part, both .3 mils / div');
               writeln(PrintDev,spc(85),TimeStamp);
```

```
                writeln(PrintDev);

writeln(PrintDev,chr(27),'A',chr(1),chr(15));    { 1/180
LPI, 17 pitch }
              end else PrinterOff;
            end;
      '0'..'9':  PosChkSpeed := (ord(key)-ord('0')) *
1000 {ms};
      'T': if PositionCheckMode = GainDrift then
PositionCheckMode := StraightXY
              else PositionCheckMode := GainDrift;
      'Q','X': begin
                 PrinterOff;
                 exit;
               end;
    end;
  until false;  { loop indefinitely }
end;
{
   ************************
   * Option 4: Beam Power *
   ************************
}
procedure Power4;
{
  as fast as possible, sound output
  have dropped tests for 65535,0 on X#,Y#
  rework sounds, can speed with pre-calculation
}
var
  code,ordkey,col,row,ScanStep,waste: Integer;
  PowerSum : array[1..3] of LongInt;
  XCorner,Ycorner,TotalWeight: LongInt;
  AvgArray : Array [1..PowerMvAvg] of LongInt;
  AvgIndex,AvgCount : Byte;
  ADC : real;
label
```

```
  Power4a,Power4b,Power4c;
begin
  gotoxy(1,25);
  write('(4) POWER: Press ',CalibSelect[NumSensors],
    ' to select sensor #, Q to Quit');
  Sensor := 1;
Power4a:
  AvgCount:=0;
  AvgIndex := 0;
  PowerSum[Sensor] := 0;
  ScanStep := 4;   { 1.4 mil = matches that used for
calib }
  Xcorner := trunc(Xsp[Sensor] - ((ColMax+1)/2.0) *
ScanStep);
  Ycorner := trunc(Ysp[Sensor] - ((RowMax+1)/2.0) *
ScanStep);

SetupBeamProfile(BeamProfileVariables,Sensor,-1,-1,-1,-1
);
  ClearLine(24);
Power4b:
      Display := false;
      GoodRead(1);
      if TotWtAvg=0 then begin
        Center(' ERROR: Sensor #'+IntStr(Sensor)+
          ' was not found: Try (6) Move or (7)
Search',24);
        WaitKey;
        exit;
      end;
      if GoodCount > 0 then begin
        Xsp[Sensor] := trunc(Xaverage+0.5);
        Ysp[Sensor] := trunc(Yaverage+0.5);
      end;
  if AvgIndex=PowerMvAvg then AvgIndex:=1
  else Inc(AvgIndex);
  if AvgCount<PowerMvAvg then Inc(AvgCount)
```

```
  else Dec(PowerSum[Sensor],AvgArray[AvgIndex]);
  AvgArray[AvgIndex]:=trunc(Power[Sensor]+0.5);
  Inc(PowerSum[Sensor],trunc(Power[Sensor]+0.5));
  gotoxy(5,20+Sensor);
  write('Sensor
#',Sensor,(PowerSum[Sensor]/AvgCount):6:2,
    ' mW Averaged Reading; (',Power[Sensor]:6:2,
    ' mW Unfiltered Reading)');
  if SoundOn then begin
    Sound(1000);
    delay(80);
    if TotalWeight<1 then Sound(37)
    else Sound(trunc(30*ln(TotalWeight)+37));
    delay(200);
  end;
{
  If 10 * round then recheck centers with goodread &
Xsp[]=XAvg
    range, duration, need time
}
Power4c:
  if not keypressed then goto Power4b
  else begin
    ordkey := IntVal(ReadKey,code);
    if (ordkey > 0) and (ordkey <= NumSensors) then
begin
      Sensor := ordkey;
      PROFMOVE := true;
      MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
        FMOVE := false;
        o Power4a;
    end;
  end;
end;
{
   ************************
   * Option 5: Beam Focus *
```

```
   ************************
}
procedure Focus5;
var
  code,ordkey,x,i,i2,stp: Integer;
  LocRowMax,cwi,rwi,HiPoints,col,row,Try: Integer;
  cwmax,rwmax,ratio,Xaverage,Yaverage,ADC: Real;
  tw,cw,rw,cplo,cphi,rplo,rphi: Real;
  TotalWeight,Xcorner,Ycorner: LongInt;
  FocusColWeight,FocusRowWeight: array[1..30] of Real;
  lnResult : real;
label
  Focus5a;
begin
  Sensor := Focus5Sensor;
{
  assumes 22x22 array for now
  1/e^2 but acting on area-integrated power
}
  SetScreen;
Focus5a:
  gotoxy(1,25);
  write('(5) Focus Parameters:  Press
',CalibSelect[NumSensors],
    'for sensor #, S to search, Q to Quit');
  with BeamProfileVariables[Sensor] do begin
    Display := false;

SetupBeamProfile(BeamProfileVariables,Sensor,Xsp[Sensor]
,Ysp[Sensor],
      4(ScanStep),trunc(Bk[Sensor] + 0.5));
    GetBeamProfile(BeamProfileVariables,Sensor);

    if (TotalWeight <> 0) and (XYWeight <> 0) then begin

      Xaverage := Xweight / XYWeight;
      Yaverage := Yweight / XYWeight;
```

```
      TotWtAvg := TotalWeight - BkgndThreshold;
      Xsp[Sensor] := trunc(Xaverage+0.5);
      Ysp[Sensor] := trunc(Yaverage+0.5);
 {    gotoxy(1,24);
      write('Xave=',Xaverage:7:1,' Yave=',Yaverage:7:1);
      gotoxy(1,25);
      write('XYweight=',XYweight,'
TotWtAvg=',TotWtAvg:7:1);
      delay(4000); }
      window(1,1,80,25);
      textmode(C80); {ClearScreen}
      SetScreen;
   if OpMode <> TextOp then begin
      gotoxy(11,3);
      write('X');
      gotoxy(39,3);
      write('Y');
      SetFillStyle(InterleaveFill,1);
{*** Graph cure depth:
     ln(Row or Col reading - appropriate background) =
lnResult;
     if lnResult <= 0 then graph two units in Y (always
5 in X);
     otherwise: multiply by lambda to get mils cure  =
milsCure;
          then: take milsCure to scanstep units by 4
bits steps/3.556 bits/mil;
               or milsCure*(4/3.556)* 7 pixels/graphed
unit
                                    = result to be
graphed. ***}

      for i := (1+15) to (ColMax+15) do begin
                          {
rectangle(0,(i-1)*4,ColWeight[i]*2,i*4); }
         lnResult := ColWeight[i-15] - (ColMax *
AveBackground);
```

```
            if lnResult <= 0 then Hist2D[3,2] := 2
            else begin
               lnResult := ln(lnResult);
               if lnResult < 0.5 then Hist2D[3,2] := 12
               else begin
                  lnResult := lnResult * lamda * (4/3.556)
* 7 /2{keep on screen};
                  Hist2D[3,2] := trunc(lnResult + 0.5);
               end;
            end;
                              Hist2D[1,2] := 0;
                              Hist2D[1,1] := (i-1)*7;
                              Hist2D[2,1] := i*7;
                              Hist2D[2,2] := 0;
                              Hist2D[3,1] := Hist2D[2,1];
                              Hist2D[4,1] := Hist2D[1,1];
                              Hist2D[4,2] := Hist2D[3,2];;
                              FillPoly(4,Hist2D);
      end;
      for i := (ColMax+1+25) to (1+ColMax+RowMax+25) do
begin
                           {
rectangle(0,(i-1)*4,RowWeight[i-ColMax-10]*2,i*4); }
         lnResult := RowWeight[i-ColMax-1-25] - (RowMax
* AveBackground);
         if lnResult <= 0 then Hist2D[3,2] := 2
         else begin
            lnResult := ln(lnResult);
            if lnResult < 0.5 then Hist2D[3,2] := 12
            else begin
               lnResult := lnResult * lamda * (4/3.556)
* 7 /2{keep on screen};
               Hist2D[3,2] := trunc(lnResult + 0.5);
            end;
         end;
                              Hist2D[1,2] := 0;
                              Hist2D[1,1] := (i-1)*7;
```

```
                                Hist2D[2,1] := i*7;
                                Hist2D[2,2] := 0;
                                Hist2D[3,1] := Hist2D[2,1];
                                Hist2D[4,1] := Hist2D[1,1];
                                Hist2D[4,2] := Hist2D[3,2];;
                                FillPoly(4,Hist2D);
      end;
   end;
end; { not TextOp }
{
  Now assume only ColMax=RowMax (=22)
}
    tw := TotalWeight - SensorBkgnd[Sensor];
    if tw <= 0 then begin
      tw := 1;
      Center('BEAM NOT FOUND',24);
      WaitKey;
      Display := true;
      BeamFound := FindHoles(Sensor);    { scans for
2-mil hole for that sensor }
      if AbortFlag then exit;
    end;

    if OpMode = TextOp then SetScreen;
    LocRowMax := ColMax;
    if RowMax > ColMax then LocRowMax := RowMax;
    cwmax := 0;
    if OpMode = TextOp then stp := 2 else stp := 1;
{    gotoxy(79,1);
    write('X'); }
    for i:=1 to LocRowMax do begin
      cw := ColWeight[i];
      cw := cw - SensorBkgnd[Sensor]/ColMax;
      if cw < 1 then cw := 0.5;
      cw := 6*(ln(cw/tw)+8);    { from 2 to 39
.5/(22*22*220) to 1 }
      if cwmax < cw then cwmax := cw;
```

```
        FocusColWeight[i] := cw;
        cwi := trunc(cw*16);    { *1.02*lamda# ? }
        if cwi < 1 then cwi := 1
        else if cwi > 79*8 then cwi := 79*8;
        i2 := i*7;
        if i mod stp = 0 then begin
          if OpMode = TextOp then begin
            gotoxy(1,i div stp+1);
            write(many('-',cwi div 8+1),'*');
      {   end else begin
            Circle(3+cwi,i2,2);
            SetLineStyle(DottedLn,0,NormWidth);
            Line(0,i2,cwi,i2); }
          end;
        end;
      end; {for i}

      cwmax := cwmax - 6;    { to find 1/e^2 of beam }
      x := trunc(cwmax * 16);
      if x < 1 then x := 1;
      if x > 79*8 then x := 79*8;
      if OpMode = TextOp then begin
        gotoxy(X div 8,12);
        write('^');
      end; { else Line (x,0,x,160);
      gotoxy(79,13);
      write('Y');  }
      rwmax := 0;
      for i:=1 to LocRowMax do begin
        rw := RowWeight[i];
        rw := rw - SensorBkgnd[Sensor]/RowMax;
        if rw < 1 then rw := 0.5;
        rw := 6*(ln(rw/tw)+8);  { tw# really not wanted
here, want absolute }
        if rwmax < rw then rwmax := rw;
        FocusRowWeight[i] := rw;
        rwi := trunc(rw*16);    { *1.02*lamda# ? }
```

```
        if rw < 1 then rw := 1;
        if rw > 79*8 then rw := 79*8;
        i2 := 155+i*7+7;
        if i mod stp = 0 then begin
          if OpMode = TextOp then begin
            gotoxy(1,12+i div stp);
            write(many('-',rwi div 8+1),'*');
        { end else begin
            Circle(3+rwi,i2,2);
            Line(0,i2,rwi,i2); }
          end;
        end;
      end; {for i}
      rwmax := rwmax - 6;
      x := trunc(rwmax * 16);
      if x < 1 then x := 1;
      if x > 79*8 then x := 79*8;
      if OpMode = TextOp then begin
        gotoxy(x div 8,23);
        write('^');
      end; { else Line(x,165,x,320);  }
      for i:=1 to LocRowMax-1 do begin
        if (FocusColWeight[i+1] > cwmax) and
    (FocusColWeight[i] < cwmax) then
          cplo := i + (cwmax - FocusColWeight[i]) /
            (FocusColWeight[i+1] - FocusColWeight[i]);
        if (FocusColWeight[i+1] < cwmax) and
    (FocusColWeight[i] > cwmax) then
          cphi := i + (cwmax - FocusColWeight[i]) /
            (FocusColWeight[i+1] - FocusColWeight[i]);
        if (FocusRowWeight[i+1] > rwmax) and
    (FocusRowWeight[i] < rwmax) then
          rplo := i + (rwmax - FocusRowWeight[i]) /
            (FocusRowWeight[i+1] - FocusRowWeight[i]);
        if (FocusRowWeight[i+1] < rwmax) and
    (FocusRowWeight[i] > rwmax) then
          rphi := i + (rwmax - FocusRowWeight[i]) /
```

```
            (FocusRowWeight[i+1] - FocusRowWeight[i]);
        end;
        gotoxy(1,24);
        write(Blank);
        gotoxy(15,24);
        write('Sensor #',Sensor,' Average Width ');
        write((cphi-cplo+rphi-rplo)*4/(2*3.556):4:1);
        if abs(rphi-rplo)<1E-5 then ratio := 0
          else ratio := (cphi-cplo)/(rphi-rplo);
        write(' mils   X/Y = ',ratio:4:2);

        if not keypressed then goto Focus5a
        else begin
          key := upcase(ReadKey);
          if key='S' then begin
            Xsp[Sensor] := Xpos;
            Ysp[Sensor] := Ypos;
            Display := true;
            if FindHoles2(Sensor) then begin;   { search
around new center }
              Xsp[Sensor] := FindBeamVariables[Sensor].Xval;

              Ysp[Sensor] := FindBeamVariables[Sensor].Yval;

              Xpos := trunc(Xsp[Sensor]);
              Ypos := trunc(Ysp[Sensor]);
            end;
            goto Focus5a;
          end else begin
            ordkey := IntVal(key,code);
            if (ordkey > 0) and (ordkey <= NumSensors) then
begin
              Sensor := ordkey;
              PROFMOVE := true;

MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
              PROFMOVE := false;
```

```
            delay(200);
            goto Focus5a;
          end;
        end;
      end;
    end;
end;
{
   ************************
   * Option 6: Move Beam *
   ************************
}
procedure Move6;
var
  KeyVal,code,i: Integer;
  InputLine,indent: String;
  ADC : real;
label
  Move6a,Move6b,Move6c;
begin
  Xpos := XRest;
  Ypos := YRest;
  Sensor := 1;
Move6a:
  TextMode(C80);
  gotoxy(1,25);
  write('(6) MOVE BEAM:');
  indent := many(' ',19);
  gotoxy(1,1);
  writeln(indent,'Sensor    Default       Last Found
Power');
  writeln(indent,'          Position       Position
(mW)');
  writeln(indent,'------- ------------ ------------
-------');
  gotoxy(1,5+NumSensors);
  write(indent,'Beam Rest Position: ');
```

```
  PrintXY(XRest,YRest);
  Center('KeyPress   Action
         ',17);
  Center('--------   ------
         ',18);
  Center(' '+CalibSelect[NumSensors] +
    '    Change sensor #
',19);
  Center('   X       Reposition X coord. of beam
         ',20);
  Center('   Y       Reposition Y coord. of beam
         ',21);
  Center('   S       Search for hole of sensor from
current position',22);
  Center('   Q       Quit Move Beam function
         ',23);
  Center('****************************************',11);
  Center('*                                      *',12);
   Center('*                                    *',13);
  Center('****************************************',14);
  for i:=1 to NumSensors do begin
    gotoxy(23,3+i);
    write(i,'    ');
    PrintXY(XSensorPos[i],YSensorPos[i]);
    write(' ');
    PrintXY(Xsp[i],Ysp[i]);
    write(' ');
    write(Power[i]:5:1);
  end;
Move6b:
  ClearLine(15);
  MoveBeam(Xpos,Ypos);
Move6c:
  Pause(30000);
  ADC := ReadLaserBoard(SensorChannel[Sensor]);
  gotoxy(28,12);
  write('Sensor #',Sensor,' Power: ',ADC:3:2);
```

```
  gotoxy(21,13);
  write('Current Beam Position: ');
  PrintXY(Xpos,Ypos);
  if SoundOn and (ADC > Bk[Sensor]) then begin
    Sound(1000);
    delay(10);
    Sound(30*trunc(ADC)+37);
    delay(20);
  end;
  case upcase(key) of
    ' ': goto Move6c;
    'Q': exit;
    'X': begin
           gotoxy(30,15);
           write('New X Position =');
           readln(InputLine);
           if InputLine<>'' then Xpos :=
WordVal(InputLine,code);
           goto Move6b;
         end;
    'Y': begin
           gotoxy(30,15);
           write('New Y Position =');
           readln(InputLine);
           if InputLine<>'' then Ypos :=
WordVal(InputLine,code);
           goto Move6b;
         end;
    'S': begin
           Xsp[Sensor] := Xpos;
           Ysp[Sensor] := Ypos;
           SetScreen;
           Display := true;
           if FindHoles(Sensor) then begin   { start
again at this new center }
             Xsp[Sensor] :=
FindBeamVariables[Sensor].Xval;
```

```
              Ysp[Sensor] :=
FindBeamVariables[Sensor].Yval;
              Xpos := trunc(Xsp[Sensor]);
              Ypos := trunc(Ysp[Sensor]);
            end;
            if AbortFlag then WaitKey;
            goto Move6a;
          end;
      else begin
        KeyVal := IntVal(key,code);
        if (KeyVal > 0) and (KeyVal <= NumSensors) then
begin
          Sensor := KeyVal;
          goto Move6b;
        end;
      end;
    end;
    goto Move6c;
end;
{
   ***************************
   * Option 7: Find Pinholes *
   ***************************
}
procedure Search7;
begin
  SetScreen;
  for i := 1 to 10 do AverageBeamPowerArray[i] := 0.0;
  { Each time find pinholes is run this array is reset;
    this way option 1 (find beam profile/power) will
always have
    two values, assuming the holes are found, with which
    to begin its average. }
  Sensor := 1;
  Display := true;
  BeamFound := FindHoles(Sensor);    { scans for 2-mil
hole around original center }
```

```
  if not AbortFlag then Pause(20000);
  PROFMOVE := true;
  MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
  PROFMOVE := false;
  if not AbortFlag then Pause(20000);
  Sensor := 2;
  BeamFound := FindHoles(Sensor);
  if not AbortFlag then Pause(20000);
end;
{$i beamcali.inc}
{-------------------------------------------------}
procedure ShowTitle;
begin
  TextMode(C80);
  SingleBar := many(#196,26);
  centerln('BEAM, Laser Beam Analysis Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  centerln(SingleBar);
end;
{
  Display Beam's Main Menu
}
procedure DisplayMenu;
var
  i: Integer;
begin
  Calibrate := false;
  ShowTitle;
  window(25,7,80,18);
  gotoxy(1,1);
  writeln('1. Display Profile');
  writeln('2. Beam Power/Profile History');
  writeln('3. Position Check');
  writeln('4. Beam Power');
  writeln('5. Beam Focus');
```

```
  writeln('6. Move Beam');
  writeln('7. Find Pinholes');
  writeln('8. Calibration Operations');
  writeln('T. Toggle between Screen Modes');
  writeln('X. Exit - Return to Main Menu');
  window(1,1,80,25);
  gotoxy(29,18);
  if OpMode = TextOp then write('  Text Display Mode  ')
    else write('Graphics Display Mode');
  gotoxy(14,20);
  write('Sensor #  Default Location      Current
Power(mW)');
  for i:=1 to 3 do begin
    gotoxy(17,20+i);
    write(i);
    gotoxy(25,20+i);
    PrintXY(XSensorPos[i],YSensorPos[i]);
    gotoxy(43,20+i);
    PrintXY(Xsp[i],Ysp[i]);
    gotoxy(59,20+i);
    write(Power[i]:5:1);
  end;
end;
{
  print option line and get selection
}
procedure GetSelection;
var
  SelectionMade: Boolean;
  Resist,Temper: Real;
  code: Integer;
begin
  ReadTemperature(Resist,Temper);
  gotoxy(1,25);
  write('MENU: SELECT OPTION      ');
  gotoxy(66,25);
  write('Temp: ');
```

```
  if Temper > 70 then begin
    write(' > 70 C');
    beep;
  end else write(Temper:5:1,' C');
  SelectionMade := false;
  repeat
    key := upcase(ReadKey);
    if (key in ['1'..'8','T','X']) or (key='T') or
(key='X') then begin
      SelectionMade := true;
      if key='T' then if OpMode = TextOp then OpMode :=
GraphicsOp
        else OpMode := TextOp;
      if key='X' then Select := 9
        else Select := IntVal(key,code);
    end;
  until SelectionMade;
end;
procedure PowerControl(bit: Integer;Op: string);
begin
  if Op = 'On' then
    port[PowerPort + 2] := port[PowerPort + 2] or bit
  else
    port[PowerPort + 2] := port[PowerPort + 2] and
($FF-bit);
end;
procedure HardReset;
begin
  port[PowerPort + 3] := $98;    { init 8255 chip }
  port[PowerPort + 3] := $98;
end;
procedure InitPowerPort;
begin
  PowerControl(1,'Off');
  if port[PowerPort + 2] and _INIT_IN > 0 then HardReset
  else begin
    PowerControl(1,'On');
```

```
    if port[PowerPort + 2] and _INIT_IN = 0 then
HardReset;
  end;
end;
procedure BeamAnalysis;
var
  i: Integer;
begin
  IOSetup;
  for i:=1 to 3 do begin
    Xsp[i] := XSensorPos[i];
    Ysp[i] := YSensorPos[i];
  end;
  oldX := XRest;   { initialize beam position }
  oldY := YRest;
  AxisMove(oldX,1);   { first time is not slew rate
limited }
  AxisMove(oldY,1);
  Search7;            { always find the beam sensor
positions }
{
  Main Program Loop
}
  repeat
    Xpos := XRest;   { Move beam to rest position }
    Ypos := YRest;
    MoveBeam(Xpos,Ypos);
    DisplayMenu;
    GetSelection;
    ClrScr;
    case Select of
      1: Display1;
      2: History2;
      3: Position3;
      4: Power4;
      5: Focus5;
      6: Move6;
```

```
      7: Search7;
      8: CaliMenu;
      9: halt(0);
    end;
  until false;
end;
procedure MakeSureLaserIsOn;
var
  Text: String;
begin
{*** commented out for Release 3.1 ***  if not
HardwareLaserOn then begin
    Exec('C:\COMMAND.COM','/C SERVICE LASER ON
NOW/i/t');
    Center('--- Waiting for Laser to Power Up ---',23);
    CursorOff;
    minu := 15;
    sec  := 0;
    key  := ' ';
    while (minu >= 0) and (key <> 'C') do begin
      while (sec >= 0) and (key <> 'C') do begin
        Text := PadSp(minu) + ':' + LZero(sec);
        gotoxy(37,16);
        write(Text);
        delay(1000);
        dec(sec);
        if keypressed then key := upcase(ReadKey);
      end;
      dec(minu);
      sec := 59;
    end;
  end; ***}
end;
{  -   M  A  I  N     P  R  O  G  R  A  M   -  -  -  -  }
 begin  { Beam }
  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);    {
clear variables }
```

```
  OldExitProc := ExitProc;
  ExitProc := @ExitBeam;
  CheckBreak := true;
  ShowTitle;
  OpMode := GraphicsOp;
  Display1Sensor := 1;
  Focus5Sensor   := 2;
  PROFMOVE := false;
  PositionCheckMode  := GainDrift;
  writeln;
  writeln;
  OutFileName := '';
  ReadBeamParamsFile;
  ReadCommandLine(CmdLine);
  CheckForOptions(CmdLine);
  MakeSureLaserIsOn;
  InitPowerPort;
  BeamAnalysis;
  QuietSystemError('Success.');
end.
```

```
{////////////////////////////////////////////////////
//                MicroStar Version 1.01A
//
//                File Name : MS.PAS
//
////////////////////////////////////////////////////}
{$C-,I-,K-}
program MicroStar;
{
        Copyright (c) 1985,86 by Borland International,
Inc.
         List of data files:
          EDITERR.EDG    - Text file containing error
messages.
}
 {$I VARS.ED     } { Constant, type, and variable
declarations }
 procedure userStatusline(Var TWindow:Byte;
                          Column,line:Integer;
                          Insertflag:Insflag;
                          WW,AI: boolean); forward;
procedure usererror(Var Msgno:Byte);forward;
procedure pulldownmenu;forward;
procedure ClearSubMenu(ScreenPosition,Width:integer);
forward;
 procedure printnext;forward;
function ErrorCheck (ErrorNum:byte) : boolean; forward;
 {$I USER.ED     } {Editor kernel and primitive level
helper routines }
{$I SCREEN.ED   } {Screen updating routines }
{$I FASTCMD.ED } { Misc. command processing routines
that are not }
                 { overlaid because they are commonly
used and must be fast }
 function ErrorCheck {(ErrorNum:byte) : boolean};
{  If the ErrorNum <> 0 then  there has been some kind
of error. The
```

```
   appropriate error message is displayed, and the user
must press escape
   before the program will resume execution.
}
var
  ch    : char;
  st    : VarString;
  count : byte;
begin { function ErrorCheck }
  IOsave := ErrorNum;                    { assign the
      error number to the    }
  if IOsave <> 0 then                    { global
      variable IOsave }
  begin
    MenuColor := PLowColor;              { Update the
      text color }
    Beep;                                { Beep to get
      the user's attention }
    SetMemAddress(1,1);                  { set the
      display coordinates to 1,1}
    st := EditMessage(IOSave);           { Get the
      appropriate error message }
    st := st + ' - Press ESC ';
    SaveScreen(SaveCmdLine,80);          { save the
      current command line text}
    WriteString(st);                     { display the
      error message }
    for count := (EndPosition - 2) downto byte(St[0]) do
      { pad line w/ spaces}
      WriteChar(32);
    repeat                               { read the
      keyboard for ESCAPE }
      Read(kbd,ch)
    until (ch = #27) and not keypressed;
    SetMemAddress(1,1);                  { set the
      display coordinates to 1,1}
    RestoreScreen(SaveCmdLine,80);       { restore
```

```
      original text on command  }
  end;                                        { line }
  ErrorCheck := (IOSave <> 0);
end;  { function ErrorCheck }
 {$I PRINT.ED }  {Printing routines}
 procedure UserReplace(Var Ch:byte);
{  First a window is created, then the string is
displayed in the window
   and the user is prompted for a character
}
const
  prompt : string[13] = 'Replace (Y/N):';
begin { procedure UserReplace }
  ch := byte (BoolPrompt (10,prompt));
end;  { procedure UserReplace }
 {********** The overlay area starts here **********}
{$I INIT.ED      }  { Overlaid version of initialization
      code }
{$I KCMD.ED      }  { Overlaid version of Ctrl-K routines
}
{$I OCMD.ED      }  { Overlaid version of Ctrl-O routines
}
{$I QCMD.ED      }  { Overlaid version of Ctrl-Q routines
}
{$I CMD.ED       }  { Overlaid version of general editing
commands }
{$I MSCMD.ED     }  { Overlaid MicroStar-specific
commands }
(*  not used:
{$I SPELL.ED     }  { Microstar/Lightning spelling
checker (overlaid) }
*)
{********** The overlay area ends here **********}
 procedure SetOnOff(var c; state: boolean); forward;
 procedure GetFlags;
{  Set the current text formating flags }
begin
```

```
  with Curwin^ do
  begin
    InsertDef := Not boolean (InsertFlag);
    SetOnOff(st41,InsertDef);
    AutoinDef := AI;
    SetOnOff(st43,AutoinDef);
    WordWpdef := WW;
    SetOnOff(st42,WordWpDef);
  end;
end;
 procedure CloseWindow;
{  If the second  window is open, the information in it
is deleted and
   the boolean flag indicating the windows status is set
to false.
   The F10 function key prompt is then displayed in
place of Window Two's status line.  Note that windows 2
and three are the two TEXT windows available in
MicroStar.  Window 1 is used for the status line window
and pull down menus.
}
begin { procedure CloseWindow }
  if not SecondWindow then           { can't do with one
      window }
    if ErrorCheck (121) then exit;
  CurrWindow := 2;                   { update current
      window to window two }
  EditWindowDelete(3);               { delete the text
      in window three }
  SecondWindow := false;             { update boolean
      flag }
  DispF10;                           { display F10 key
      prompt }
  GetFlags;
  UpdCurFlag := true;
end;  { procedure CloseWindow }
 procedure ToggleWindow;
```

```
{  If both windows are currently defined, the cursor is
moved from the current window to the alternate.
}
begin { procedure ToggleWindow }
  if not SecondWindow then    { can't do with one window
      }
    if ErrorCheck(121) then exit;
  if CurrWindow = 3 then      { toggle current window }
    Currwindow := 2
  else
    Currwindow := 3;
  EditWindowGoto(CurrWindow); { update current window's
cursor, etc. }
  GetFlags;
end;  { procedure ToggleWindow }

procedure OpenWindow;
{  First a window is created on the screen and a prompt
is displayed in it asking for the number of lines of the
screen to be used for the second window.
}
const
  st:String[25] = 'How many lines (3 - 17): ';
  cl = 40; ln = 7; wd = 34; hg = 3;
var
  xy : byte;
  ok : boolean;
begin { procedure OpenWindow }
  if SecondWindow then                { if 2 windows
      open, error }
    if ErrorCheck(120) then exit;
  MakeWindow(cl,ln,wd,hg);            { make prompt
      window }
  ok := GetInteger(WindowLines,3,17,cl + 2,ln +
      1,st,dummyst);
  RestoreWindow(cl,ln,wd,hg);         { restore screen
      contents }
```

```
  if ok then
  begin
    EditWindowCreate(WindowLines+1,2);      { allocate a
      text window }
    EditWindowLink(2,3);                    { link the
      new window w/ the text }
    SetMemAddress(1,25 - Windowlines);      { set display
      address on screen }
    MenuColor := PFrameColor;               { set display
      color }
    WriteChar(LConnect);                    { display
      window frame }
    for xy := 2 to 79 do
      WriteChar(Horizontal);
    WriteChar(UpperRight);
    SecondWindow := true;                   { set global
      flag to true }
    with Curwin^.Fwdlink^ do                { the new
      window is next on list  }
    begin                                   { set its
      flags like current ones }
      Insertflag := Insflag(not Insertdef);
      WW := WordWpdef;
      AI := AutoIndef;
    end;
    ToggleWindow;                           { go to it
      right away }
  end;
end;  { procedure OpenWindow }
 procedure SetLoggedDrive;
{  Sets a new logged drive/directory }
const
  cl = 1; ln =11; wd = 80; hg = 4;
  prompt : String[20] = 'New Drive/Directory:';
var
  NewFilePath : fstr66;
begin { procedure SetLoggedDrive }
```

```
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFilter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(prompt);
  MenuColor := WNormColor;
  NewFilePath := GetString(cl + 23,ln +
      1,54,FilePath,true);
  If NewFilePath = #27 then NewFilePath := '';
  if NewFilePath <> '' then
  begin
    ChDir(NewFilePath);              { try to change to
      new directory }
    if IOResult <> 0 then
    begin
      if ErrorCheck(122) then;
    end else
    begin
      GetDir(0,FilePath);            { if no errors, use
      DOS to get std form }
      if not ErrorCheck(IOResult) then;
    end;
  end;
  RestoreWindow(cl,ln,wd,hg);
end; { procedure SetLoggedDrive }
 procedure ProcessFileName(var FileName : fstr66);
{  Takes the filename and adds the directory path, etc.
      to it }
var
  i : integer;
  TmpDir : fstr66;
begin { procedure ProcessFileName }
  if FileName = '' then exit;
  if (length(FileName) = 1) or
     ((length(FileName) >= 2) and (FileName [2] <> ':'))
then
        FileName := Copy(FilePath,1,2) + FileName;
```

```
  if length(FileName) >= 3 then
    if FileName [3] <> '\' then
    begin
      GetDir(Succ(Ord(FileName[1]) - Ord('A')),TmpDir);
      if IOResult <> 0 then
      begin
        if ErrorCheck(122) then;
        FileName := '';
      end else
        begin
        if TmpDir[length(TmpDir)] <> '\' then
          TmpDir := TmpDir + '\';
        Insert(Copy(TmpDir,3,length(TmpDir) -
2),FileName,3);
        end;
    end;
end; { procedure ProcessFileName }

function WriteFile(BlockOnly: boolean): boolean;
{  Writes a block/file of text to disk }
const
  cl = 1; ln = 10; wd = 80; hg = 4;
  blockmsg: String[31] = 'Writing the block. Please wait
';
  oldfile: String[31] =  'File exists. Continue (Y/N)?
';
var
  i : byte;
  StPtr: ^fstr66;
  fl: file;
  ch: char;
  p1,p2: Plinedesc;
  SaveCursorMode: integer;
begin { procedure WriteFile }
  WriteFile := false;
  if BlockOnly then
```

```
    if BlockHide or (BlockFrom = nil) or (BlockTo = nil)
then
      if ErrorCheck (26) then exit;
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFilter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(fileprompt);
  MenuColor := WNormColor;
  if BlockOnly then
    StPtr := Addr(BlockFileName)
  else
    StPtr := Addr(WriteFileName);
  StPtr^ := GetString(cl + 12,ln + 1,66,StPtr^,true);
  If StPtr^ = #27 then StPtr^ := '';
  ProcessFileName(StPtr^);
  if StPtr^ <> '' then
  begin
    ch := 'Y';
    Assign(fl,StPtr^);
    reset(fl);
    if IOResult = 0 then
    begin
      MenuColor := WLowColor;
      SetMemAddress(cl + 2,ln + 2);
      WriteString(oldfile);
      repeat ch := Upcase(readChar); until Ch in
['Y','N', #27];
      close(fl);
    end;
    if ch = 'Y' then
    begin
      MenuColor := WLowColor;
      SetMemAddress(cl + 2, ln + 2);
      if BlockOnly then begin
```

```
          p1 := BlockFrom;
          p2 := BlockTo;
          WriteString(blockmsg);
        end
        else
        begin
          WriteString(filemsg);
          p1 := Curwin^.Topline;
          while p1^.Backlink <> nil do  { Search for
beginning of stream }
            p1 := p1^.Backlink;
          p2 := nil;                    { Last line has
nothing after }
        end;
        if EditBlockWrite(StPtr^, p1, p2) then
        begin
          WriteFile := true;
          if not BlockOnly then
          begin
            EditChangeFlag := false;
            WorkFileName := WriteFilename;
            NoWorkFile := false;
            dispFileName;
          end;
        end;
      end;
    end;
    RestoreWindow(cl,ln,wd,hg);
    SetCursorMode(SaveCursorMode);
end; { procedure WriteFile }
 function OKtoAbandon: boolean;
{  Displays the user prompt to abandon file changes or
not }
const
  prompt : String[23] = 'Abandon changes (Y/N)? ';
var
  ch:char;
```

```
begin { function OKtoAbandon }
  ch := 'Y';
  if EditChangeFlag then
    ch := BoolPrompt(10,prompt);
  OKtoAbandon := (ch = 'Y');
end; { function OKtoAbandon }
 procedure ClearFile;
{  Delete the text from the current window }
begin { procedure ClearFile }

ClearSubmenu(SubMenuPos,Mem[Cseg:MenuData[CurrSubMenu].S
ubStart]);
  if SecondWindow then CloseWindow;
  EditWindowDeleteText;
  noworkfile := true;
  EditChangeFlag := false;
  UpdCurFlag := true;
  EditUpdPhyscr;
end; { procedure ClearFile }
 function Abandonfile: boolean;
{  Abandon the file being edited }
begin
  AbandonFile := false;
  if OKtoAbandon then
  begin
    ClearFile;
    AbandonFile := true;
  end;
end; {AbandonFile}
 function Savefile : boolean;
{  Save the current file to disk }
const
  cl = 15; ln = 10; wd = 34; hg = 3;
var
  i  : Byte;
  st : fstr66;
  p  : Plinedesc;
```

```
  SaveCursorMode: integer;
begin { procedure SaveFile }
  if NoWorkFile then
  begin
    SaveFile := WriteFile(false);
    exit;
  end;
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  SaveFile := false;
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(filemsg);
  if (Pos('.BAK',WorkFilename) = 0) and FileExist then
  begin
    i := Pos('.',WorkFilename);
    if i = 0 then st := WorkFilename + '.BAK'
             else st := Copy(WorkFilename,1,i) + 'BAK';
    i := CopyFile(false,WorkFilename,st);
    if i < 0 then if errorcheck(104) then;
  end;
  p := Curwin^.Topline;
  while p^.Backlink <> nil do  { Search for beginning of
stream }
    p := p^.Backlink;
  if EditBlockWrite(WorkFilename, p, nil) then
  begin
     EditChangeFlag := false;
     SaveFile := true;
  end;
  RestoreWindow(cl,ln,wd,hg);
  SetCursorMode(SaveCursorMode);
end;  { procedure SaveFile }
 procedure ExitFile;
{  Exit the editor after checking if the user wants to
halt printing/
```

```
    save changes
  }
  const
    StopPrint : String[15] = 'Abort Printing (Y/N)?';
    SaveIt : String [19] = 'Save Changes (Y/N)?';
  var
    ch: char;
  begin { procedure ExitFile }
    ch := 'N';
    if EditChangeFlag then
      ch := boolprompt(10,SaveIt);
    if ch = #27 then exit;
    if ch = 'Y' then
     if not SaveFile then exit;
    if printing then
    begin
      ClearFile;
      if boolprompt(10,StopPrint) <> 'Y' then exit;
    end;
    printexit;
    EditExit;
    ClrScr;
    SetCursorMode(CursorOn);
    Halt;
  end;  { procedure ExitFile }
  procedure ReadBlock;
  {  Read a block into the current window }
  const
    cl = 1; ln = 10; wd = 80; hg = 4;
    proceed: String[29] = 'Reading the file. Please wait';
  var
    i : byte;
    st: fstr66;
    fl: file;
    ch: char;
    SaveCursorMode: integer;
  begin { procedure ReadBlock }
```

```
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  MakeWindow(cl,ln,wd,hg);
  MenuColor := WLowColor;
  CharFilter := FileCharFIlter;
  SetMemAddress(cl + 2, ln + 1);
  WriteString(fileprompt);
  MenuColor := WNormColor;
  BlockFilename := GetString(cl + 12,ln +
1,66,BlockFilename,true);
  If BlockFilename = #27 then BlockFilename := '';
  ProcessFileName(BlockFileName);
  if BlockFilename <> '' then
  begin
    MenuColor := WLowColor;
    SetMemAddress(cl + 2, ln + 2);
    WriteString(proceed);
    EditFileRead(BlockFilename);
  end;
  RestoreWindow(cl,ln,wd,hg);
  SetCursorMode(SaveCursorMode);
end;  { procedure ReadBlock }
procedure OpenFile(prompting: boolean);
{  Opens a new file specified by the user }
const
  cl = 1; ln = 9; wd = 80; hg = 4;
  prompt : String[10] = 'Open File:';
  proceed: String[29] = 'Reading the file. Please wait';
  newfile: String[29] = 'File is new. Continue (Y/N): ';
var
  fl: file;
  ch: char;
  SaveCursorMode: integer;
begin { procedure OpenFile }
  SaveCursorMode := GetCursorMode;
  SetCursorMode(Cursoroff);
  MakeWindow(cl,ln,wd,hg);
```

```
MenuColor := WLowColor;
CharFilter := FileCharFilter;
SetMemAddress(cl + 2, ln + 1);
WriteString(prompt);
MenuColor := WNormColor;
if prompting then
  WorkFilename := GetString(cl + 13,ln +
    1,66,WorkFilename,true)
else begin
  SetMemAddress(cl + 13, ln + 1);
  WriteString(WorkFileName);
end;
If WorkFilename = #27 then WorkFilename := '';
ProcessFileName(WorkFileName);
if WorkFilename <> '' then
begin
  ch := 'Y';
  Assign(fl,WorkFilename);
  if not ErrorCheck(IOResult) then
  begin
    reset(fl);
    if IOResult = 0 then
    begin
      close(fl);
      FileExist := true;
      MenuColor := WLowColor;
      SetMemAddress(cl + 2, ln + 2);
      WriteString(proceed);
      EditFileRead(WorkFilename);
      FileExist := true;
      NoWorkfile := false;
    end
    else
    begin
      MenuColor := WLowColor;
      SetMemAddress(cl + 2,ln + 2);
      WriteString(newfile);
```

```
          repeat ch := Upcase(readChar); until Ch in
             ['Y','N'];
          if ch = 'Y' then
          begin
            NoWorkfile := false;
            FileExist := false;
          end;
        end;
      end;
    end;
    dispFileName;
    EditchangeFlag := false;
    RestoreWindow(cl,ln,wd,hg);
    UpdCurflag := true;
    SetCursorMode(SaveCursorMode);
end;  { procedure OpenFile }
 procedure CloseFile;
{  Closes the current file after asking the user if the
changes should
   be saved.
}
const
  prompt : String[20] = 'Save changes (Y/N)? ';
var
  ch:char;
begin { procedure CloseFile }
  if EditChangeFlag then
  begin
    ch := BoolPrompt(10, prompt);
    if ch = #27 then exit;
    if ch = 'Y' then
      if not SaveFile then exit;
  end;
  ClearFile;
  OpenFile(true);
end;  { procedure CloseFile }
 Procedure ToggleInsert;
```

```
{  This routine toggles the flags indicating if the user
is in AutoInsert mode.  It also updates the menu and
screen.
}
begin { procedure ToggleInsert }
  EditToggleInsert;          { toggle window's flag }
  GetFlags;                  { update global flags  }
  EditUpdwinsl(CurWin);      { fix status line, too }
end;  { procedure ToggleInsert }
 Procedure ToggleAutoIndent;
{  This routine toggles the flags indicating if the user
is in
   AutoIndent mode.  It also updates the menu.
}
begin { procedure ToggleAutoIndent }
  EditToggleAutoindent;       { toggle window's flag }
  GetFlags;                   { update global flags }
end;  { procedure ToggleAutoIndent }
 Procedure ToggleWordwrap;
{  This routine toggles the flags indicating that the
user is in WordWrap mode.  It also updates the menu and
screen.
}
begin { procedure ToggleWordWrap }
  EditToggleWordwrap;      { toggle window's flag }
  GetFlags;                { update global flags  }
  EditUpdwinsl(CurWin);    { fix status line, too }
end;  { procedure ToggleWordWrap }
 Procedure LeftMargin;
{  Displays a window containing the prompt and reads the
input from the
   user.
}
const
  st:String[17] = 'New left margin: ';
  cl = 34;                              { window
      coordinates }
```

```
  ln = 9;
  wd = 26;
  hg = 3;
begin { procedure LeftMargin }
  MakeWindow(cl,ln,wd,hg);            { make the prompt
      window }
  if GetInteger(LMargindef,0,Rmargindef-1,cl + 2,ln +
1,st,st44) then
    EditSetLeftMargin(LMargindef);    { set new left
margin }
  RestoreWindow(cl,ln,wd,hg);         { restore original
screen }
end;  { procedure LeftMargin }

procedure RightMargin;
{  Displays a window containing the prompt and reads the
input from the
   user.
}
const
  st:String[18] = 'New right margin: ';
  cl = 34;                             { window
coordinates }
  ln = 10;
  wd = 27;
  hg = 3;
begin { procedure Rightmargin }
  MakeWindow(cl,ln,wd,hg);             { create the
prompt window }
  if GetInteger(RMargindef,LMargindef+1,MaxLinelength,cl
+ 2,ln + 1,st,st45) then
    EditSetRightMargin(RMargindef);    { set the new
      right margin }
  RestoreWindow(cl,ln,wd,hg);          { restore original
     screen  }
end;  { procedure SetRightMargin }
 procedure DefineTab;
```

```
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[14] = 'New tab size: ';
  cl = 34;                          { window coordinates }
  ln = 11;
  wd = 23;
  hg = 3;
begin { procedure DefineTab }
  MakeWindow(cl,ln,wd,hg);          { display the window }
  if GetInteger(TabDef,1,255,cl + 2,ln + 1,st,st46) then
    EditDefineTab(Tabdef);          { define new tab width
}
  RestoreWindow(cl,ln,wd,hg);       { restore original
screen }
end;  { procedure DefineTab }
 procedure SetUndoLimit;
{  Displays a window containing the prompt and reads the
input from the
   user.
}
const
  st:String[16] = 'New undo limit: ';
  cl = 34;                          { window coordinates }
  ln = 12;
  wd = 25;
  hg = 3;
begin { procedure SetUndoLimit }
  MakeWindow(cl,ln,wd,hg);          { diaplay the window }
  if GetInteger(Undodef,1,255,cl + 2,ln + 1,st,st47)
then
    EditSetUndolimit(Undodef);      { sets new undo limit}
  RestoreWindow(cl,ln,wd,hg);       { restore old screen }
end;  { procedure SetUndoLimit }
procedure GotoLine;
```

```
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[12] = 'Which line: ';
  cl = 27;                        { window coordinates }
  ln = 8;
  wd = 21;
  hg = 3;
begin { procedure GotoLine }
  MakeWindow(cl,ln,wd,hg);        { display window     }
  if GetInteger(Targetline,1,30000,cl + 2,ln +
1,st,dummyst) then
    EditGotoline(Targetline);     { goto the line      }
  RestoreWindow(cl,ln,wd,hg);     { restore screen     }
end;  { procedure GotoLine }
 procedure GotoColumn;
{  Displays a window containing the prompt and reads the
input from the user.
}
const
  st:String[14] = 'Which column: ';
  cl = 27;                         { window coordinates }
  ln = 9;
  wd = 23;
  hg = 3;
begin { procedure GotoColumn }
  MakeWindow(cl,ln,wd,hg);         { display window  }
  if GetInteger(TargetColumn,1,255,cl + 2,ln +
1,st,dummyst) then
    EditGotoColumn(TargetColumn); { goto the column }
  RestoreWindow(cl,ln,wd,hg);      { restore screen  }
end;  { procedure GotoColumn }
procedure GotoMarker;
{  Displays a window containing the prompt and reads the
inut from the user.
}
```

```
const
  st:String[14] = 'Which marker: ';
  cl = 27;                          { window coordinates }
  ln = 10;
  wd = 23;
  hg = 3;
begin { procedure GotoMarker }
  MakeWindow(cl,ln,wd,hg);          { display window }
  if GetInteger(TargetMarker,1,Maxmarker,cl + 2,ln +
1,st,dummyst) then
    EditJumpMarker(TargetMarker);{ jump to marker }
  RestoreWindow(cl,ln,wd,hg);       { restore screen }
end;  { procedure GotoMarker }
 procedure SetMarker;
{  This routine sets a text marker at the current cursor
position }
const
  st:String[15] = 'Select marker: ';
  cl = 27;                          { window coordinates }
  ln = 11;
  wd = 24;
  hg = 3;
begin { procedure SetMarker }
  MakeWindow(cl,ln,wd,hg);          { display window      }
  if GetInteger(TargetMarker,1,Maxmarker,cl + 2,ln +
1,st,dummyst) then
    EditSetMarker(TargetMarker); { define text marker }
  RestoreWindow(cl,ln,wd,hg);       { restore screen      }
end;  { procedure SetMarker }
 {$I PULLDOWN.ED}                   { Pulldown menu
routines }
 function SecondChar(st:st40):Byte;
var
  ch:Byte;
begin { function SecondChar }
  MenuColor := PLowColor;
  SetmemAddress(1,1);
```

```
  SaveScreen(SaveCmdLine,80);
  WriteString(St);
  MenuColor := PNormColor;
  for ch := EndPosition - 2 downto Byte(St[0]) do
WriteChar(32);
  SecondChar := EditCtrlchar;
  SetmemAddress(1,1);
  RestoreScreen(SaveCmdLine,80);
end;  { function SecondChar }
 { INCLUDE MICROSTAR PREFIXED COMMAND DISPATCHERS }
{$I K.ED}
{$I O.ED}
{$I Q.ED}
 procedure UserStatusline((Var TWindow:Byte;
Column,line:Integer;
                          Insertflag:Insflag; WW,AI:
boolean));
{ This routine updates the UserStatus line for the
current window }
const
  Statusline :  String[21] = 'Line:     1 Col:     1 ';
  is : String[2] = 'IN';
  nn : String[2] = '  ';
var
  st:string[5];
begin { procedure UserStatusLine }
  if TWindow <> 1 then
  begin
    SetMemAddress(58,TWindow);
    If TWindow = CurrWindow then
      MenuColor := WLowColor
    else
      MenuColor := WNormColor;
    Str(line:5,st);
    move(St[1],Statusline[6],5);
    Str(Column:5,st);
    move(St[1],Statusline[16],5);
```

```
    WriteString(StatusLine);
    if TWindow = CurrWindow then      { show flags for
      current only }
    begin
      MenuColor := WLowColor;         { set the display
           color }
      SetMemAddress(53,2);            { set address of
status area }
      if boolean (Insertflag) then    { enum type =>
false if insert }
        WriteString(nn)               { if no insert,
display blanks }
      else
        WriteString(is);              { if insert then
display IN }
      SetMemAddress(53,3);
      if WW then
        WriteString(Stww)             { if true then
display WW }
      else
        WriteString(nn);              { else display
blanks }
    end;
  end;
  TWindow := 0;
end;  { procedure UserStatusLine }

procedure Init;
{  The copyright is displayed, the workfile name is
requested, and all
   global variables, data structures, etc. are
initialized.
}
const st1:string[44] = ' EDIT, System Editor';
      st2:String[44] = ' 3D Systems SLA-1
Stereolithography System';
```

```
      st3:String[44] = ' Version 3.00              3D
Systems. Inc.';
var fl:file;
     i:Byte;
    ParmUsed : boolean;

  procedure DisplayCopyRight;
  begin { procedure DisplayCopyRight }
    MakeWindow(1,1,47,6);
    MenuColor := WNormColor;
    SetmemAddress(3,2);
    WriteString(st1);
    SetmemAddress(3,3);
    WriteString(st2);
    SetmemAddress(3,5);
    WriteString(st3);
  end;  { procedure DisplayCopyRight }
   procedure DisplayLineStatWindow;
  begin { procedure DisplayLineStatWindow }
    MakeWindow(51,1,30,4);
    SetMemAddress(56,1);
    WriteChar(TConnect);
    SetMemAddress(56,2);
    WriteChar(Vertical);
    SetMemAddress(56,3);
    WriteChar(Vertical);
    SetMemAddress(56,4);
    WriteChar(LRConnect);
  end;  { procedure DisplayLineStatWindow }
procedure OverlayError;
{  This routine causes the program to abort because an
error
   was encountered when trying to locate the overlay
file.
}
begin
  Writeln('Overlay file not found.  Program aborted.');
```

```
  halt;
end; { OverlayError }

begin { procedure Init }
  assign(fl,'\3DSYS\EDITOR\EDIT.000');    { check for
overlay file }
  if IOresult <> 0 then OverlayError;
  reset(fl);
  if IOresult <> 0 then OverlayError;
  close(fl);
  if IOresult <> 0 then OverlayError;
  FilePath := '\3DSYS\EDITOR';
  if ErrorCheck(IOresult) then halt;
  OvrPath(FilePath);
  if ErrorCheck(IOresult) then halt;
  ErrorFile := DefErrorFile;
  if FilePath[length(FilePath)] <> '\' then { same for
error message file }
    ErrorFile := '\' + ErrorFile;
  ErrorFile := FilePath + ErrorFile;
  GetDir(0,FilePath);                  { always look in
right dir }
  CursorOn := GetCursorMode;    { assume the cursor was
on when we entered   }
  EditInitialize;               { call EDITOR TOOLBOX
initialization routine }
  EditWindowCreate(22,1);
  EditWindowGoto(2);
  DisplayCopyRight;
  SecondWindow := false;
  XPosition := 1;
  YPosition := 1;
  CurrWindow := 2;
  ParmUsed := ParamCount <> 0;
  if ParmUsed then    { check if filename was entered at
command line }
  begin
```

```
    WorkFileName := ParamStr(1);
    for i := 1 to Byte(WorkFilename[0]) do
      WorkFilename[i] := Upcase(WorkFilename[i]);
  end;
  EditChangeFlag := false;
  NoWorkFile := true;
  gotoxy(1,5);                {Give cursor a reasonable
initial position}
  OpenFile(not ParmUsed);
  for i := 5 to 6 do
  begin
    SetMemAddress(1,i);
    RestoreScreen(WArray[pred(i),0],47);
  end;
  DisplayLineStatWindow;
  InitMainmenu;
  ExitPulldown;
  if boolean (Curwin^.Insertflag) = Insertdef then
ToggleInsert;
  if Curwin^.WW <> WordWpdef  then ToggleWordwrap;
  if Curwin^.AI <> AutoinDef then ToggleAutoindent;
  DispF10;
  EditSetLeftMargin(LMargindef);
  EditSetRightMargin(RMargindef);
  EditDefineTab(Tabdef);
  EditSetUndolimit(Undodef);
end;  { procedure Init }
procedure UserCommand(var ch : byte);
{  This is the user command processor hook }
begin { procedure UserCommand }
  if ch = 0 then {Function key}
     begin
     ch := EditGetInput;
     if ch = 68 then {F10 pulls down menu}
        PulldownMenu;
     ch := 255; {Always absorb keycode}
     end;
```

```
end;  { procedure UserCommand }

procedure UserError {(var Msgno : byte)};
{  This is the user error handler hook }
begin { procedure UserError }
  If ErrorCheck(Msgno) then;
  Msgno := 0;
end;  { procedure UserError }
procedure UserTask;
{  This is the user BackGround task hook }
begin { procedure UserTask }
  printnext;
end;  { procedure UserTask }

{$I DISP.ED   }       { MicroStar main command
dispatcher          }
{$I TASK.ED   }       {scheduling subsystem and central
dispatcher }
{$I INPUT.ED  }       {input routines
         }

begin  { main program block }
  Init;
  EditSystem;  { Use the default main loop }
  ClrScr
end.
^Z
```

```
{================================================
   3D Systems Stereolithography System Software
 ================================================
    POWER.PAS
   SLA-1 Power Control Program
  ------------------------------------------------

   Recent History:
   11/12/87  Ver 2.30  first Beta software release
    1/08/88  Ver 2.50  retitled program
    1/25/88  Ver 2.60  better messages
    2/08/88  -         converted from Turbo Basic to
                       Turbo Pascal Version 4.0
                       better Laser Board I/O
                       power status line display
                       support of toggling-Laser power
    2/26/88  Ver 2.62  fix for toggling Laser power
    3/09/88  Ver 2.80  added copyright notice
    4/04/88  Ver 3.00  change menus
    5/08/88  Ver 3.10  add laser timer code
    7/18/88  Ver 3.20  update VersionId
    9/01/88  Ver 3.21  issue service laser on/off now
                       for on/off
  ------------------------------------------------}
{$M 16384,0,0}

uses Crt,Dos;

const
  VersionId = '3.21';
  LaserBoardAddress = $300;
  PowerPort     = $304;
  _INIT_OUT     = $01;  { outputs }
  _ZSTAGE_OUT   = $02;
  _SHUTTER_OUT  = $04;
  _LASER_OUT    = $08;
  _INIT_IN      = $10;  { inputs }
```

```
  _DOOR_IN      = $20;
  _SHUTTER_IN   = $40;
  _LASER_IN     = $80;
  Indent = 22;
  LASERVECTOR  = $64;
  STEREOVECTOR = $65;
  HomeDirectory = 'C:\3DSYS\';
  FixFileName   = 'FIX.DAT';
  ParamFileName = 'POWER.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  Ctrl C = ^C;
  Zel: array[1..12] of Integer =
    (1,4,3,6,1,4,6,2,5,0,3,5);
  DazeType: array[0..6] of String =
    ('Sunday','Monday','Tuesday','Wednesday','Thursday',
    'Friday','Saturday');
  DayOfWeek : String[21] = 'SunMonTueWedThuFriSat';

type
  str 4 = string[4];
  str16 = string[16];
  str10 = string[10];
  str80 = string[80];
  LaserPowerType = (_Unknown,_Level,_Toggle);
  OnOffType = (On,Off);

var
  key: Char;
  regs: Registers;
  ParamFile,ErrorFile,FixFile : text;
  LaserVersion,StereoVersion: str4;
  CurrentDate, CurrentTime : str80;
  temp,IndentStr,FixFileLine,ParamFileLine: String;
  i,IOerr: Integer;
  sec,minu,hour,day,mon,year,weekday : Integer;
  QSize  : LongInt;
```

```
  LaserPower: LaserPowerType;
  LaserOff,ShowStuff: Boolean;

{
   Cursor on / off routines
}

function HexByte(num:Byte):str4;
const
   HexChars: str16= '0123456789ABCDEF';
begin
   HexByte := HexChars[num shr 4 + 1] + HexChars[num
    and $0F +1];
end;

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
```

```
end;
 procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln('ERROR: '+Text);
  beep;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'POWER: ',Text);
  close(ErrorFile);
end;

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
begin
  if whereX>1 then writeln;
  writeln('*** ',Text);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
```

```
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);

  IntStr := temp;
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
```

```
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = Ctrl C then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;

procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
   integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                   {  Type          1    2    3    4
}
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
```

```
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;

{$i lctrl.inc}

procedure ShowPowerStatus;
begin
  gotoxy(12,24);
  case LaserPower of
    _Unknown: write('U');
    _Level  : write('L');
    _Toggle : write('T');
  end;
  write('-Laser ');
  if port[PowerPort + 2] and _LASER_IN   = 0 then
write('On ')
    else write('Off');
  write('   Shutter ');
  if port[PowerPort + 2] and _SHUTTER_IN = 0 then
write('Open  ')
    else write('Closed');
```

```
  write('   Door ');
  if port[PowerPort + 2] and _DOOR_IN    = 0 then
write('Open  ')
    else write('Closed');
  write('   Elevator ');
  if port[PowerPort + 2] and _ZSTAGE_OUT = 0 then
write('Off')
    else write('On ');
end;

procedure SendLaserCommand(cmd: String);
begin
  ShowError(STEREO(cmd));
end;

procedure PowerControl(bit: Integer;Op: OnOffType);
begin
  if Op = On then
    port[PowerPort + 2] := port[PowerPort + 2] or bit
  else
    port[PowerPort + 2] := port[PowerPort + 2] and
($FF-bit);
end;

procedure ToggleLaserPowerOn;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN = 0) or
(loop1 > 100000);
  PowerControl(_LASER_OUT,Off);
```

```
  if (loop1 > 100000) then
    PrintError('Can''t detect laser toggle feedback.');
end;

procedure ToggleLaserPowerOff;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN > 0) or
(loop1 > 100000);
  PowerControl(_LASER_OUT,Off);
  if (loop1 > 100000) then
    PrintError('Can''t detect laser toggle feedback.');
end;

procedure TurnOnLaser;
var
  loop: LongInt;
begin
  if LaserPower = _Toggle then begin
    if port[PowerPort + 2] and _LASER_IN > 0 then
ToggleLaserPowerOn;
  end else PowerControl(_LASER_OUT,On);
  if LaserPresent and StereoPresent then begin
    SendLaserCommand('HR;CL;MD BL +');
    assign(FixFile,FixFileName);
    {$I-}
    reset(FixFile);
    IOerr := IOresult;
    if IOerr <> 0 then begin
      assign(FixFile,HomeDirectory + FixFileName);
      reset(FixFile);
      IOerr := IOresult;
```

```
    end;
    {$I+}
    if IOerr = 0 then begin
      while not eof(FixFile) do begin
        readln(FixFile,FixFileLine);
        if (FixFileLine <> '') and (FixFileLine[1] <>
'!') then
          SendLaserCommand(FixFileLine);
      end;
      close(FixFile);
    end;
  end;
end;

procedure TurnOnElevator;
begin
  PowerControl(_ZSTAGE_OUT,On);
end;

procedure OpenShutter;
begin
  PowerControl(_SHUTTER_OUT,On);
end;

procedure TurnOffLaser;
begin
  gotoxy(18,17);
  write('--- WARNING ---');
  gotoxy(18,18);
  write('The laser may take half an hour to restart!');
  gotoxy(18,19);
  write('Are you sure you want to turn the laser off?
');
  repeat
    ShowPowerStatus;
  until keypressed;
  HandleKeyboardRequest;
```

```
  if upcase(key)='Y' then
    if LaserPower = _Toggle then begin
      if port[PowerPort + 2] and _LASER_IN = 0 then
ToggleLaserPowerOff;
    end else PowerControl(_LASER_OUT,Off);
end;

procedure TurnOffElevator;
begin
  PowerControl(_ZSTAGE_OUT,Off);
end;

procedure CloseShutter;
begin
  PowerControl(_SHUTTER_OUT,Off);
end;

procedure GetSelection;
begin
  repeat
    repeat
      ShowPowerStatus;
    until keypressed;
    HandleKeyboardRequest;
    key := upcase(key);
    if key in ['1'..'7','X'] then exit;
  until false;
end;

procedure ShowPowerMenu;
begin
  window(1,1,80,25);
  ClrScr;
  centerln('POWER, SLA-1 Power Control Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
```

```
  centerln('3D Systems Laser Stereolithography System');

  for i:=1 to 5 do writeln;
  DrawBox(Indent-1,8,80-Indent,17,1);
  window(Indent,9,80-Indent-1,16);
  ClrScr;
  writeln(' 1. Turn Laser and Mirror Drive On');
  writeln(' 2. Turn Elevator On');
  writeln(' 3. Open Shutter');
  writeln(' 4. Turn Laser and Mirror Drive Off');
  writeln(' 5. Turn Elevator Off');
  writeln(' 6. Close Shutter');
  writeln(' 7. Set Laser On/Off Time');
  write(' X. Exit');
  window(1,1,80,25);
  gotoxy(28,19);
  write('Please make a selection.');
  DrawBox(10,23,69,25,1);
end;

procedure UserWait;
begin
   writeln;
   writeln;
   writeln('                ... press any key to
continue ');
   repeat delay(100) until KeyPressed;
   key:=Readkey;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
```

```
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

function Zeller(month,day,year:integer):Integer;
begin
   if month<3 then dec(year);
   Zeller := ( Zel[month] + day + year + (year shr 2))
mod 7;
end;

procedure SetLaserOnOffTime;
  var
     ChronoFile  :  text;
     ChronoStr,ServiceStr   :  String;
     AbsDate     :  array[1..8] of char;
     AbsTime     :  array[1..8] of char;
     OffsetHour, OffsetMonth  :  array[1..2] of char;
     yearHold,nnnnstr    :  string[4];
     OffsetDay   :  array[1..3] of char;
     Fileexists, AM, fileempty, endoffile  : boolean;
     OnEntryExists, OffEntryExists : boolean;
```

```
    TryToAddOn, TryToAddOff       : boolean;
    j, nnn, valcode          :  integer;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetChipTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;   { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;    { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;    { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;    { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
```

```
                port[$70] := $09;     { get year }
                Year := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then good :=
true;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
  Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
    from BCD }
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
  WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetChipTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetChipTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
{  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear; }
```

```
  CurrentDate := DazeType[Zeller(Mon,Day,Year)]+'
'+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

 procedure ShowCurrentLDT;
begin
     ClrScr;
     write('      Current Laser Status = ');
     if port[PowerPort + 2] and _LASER_IN   = 0 then
writeln('On ')
     else writeln('Off');
     GetChipTimeStamp;
     writeln('      Current Date         =
',CurrentDate);
     writeln('      Current Time         =
',CurrentTime);
     writeln;
end;

function DayOfWeekZeller:integer;
var
```

```
   imonth, iday, iyear : integer;
   cmonth, cday, cyear : array[1..2] of char;
begin
   for i:= 1 to 2 do begin
      cmonth[i] := AbsDate[i];
      cday[i]   := AbsDate[i+3];
      cyear[i]  := AbsDate[i+6];
   end;
   val(cmonth,imonth,valcode);
   val(cday,iday,valcode);
   val(cyear,iyear,valcode);
   DayOfWeekZeller:= Zeller(imonth,iday,iyear);
end;

 procedure IssueTimeCommand(laserCMD: str4);
   var
      CMDnoperiod : str4;
      iend        : integer;
begin
    if laserCMD = 'off.' then TryToAddOff:=true
                         else TryToAddOn :=true;
    CMDnoperiod:=laserCMD;
    CMDnoperiod[pos('.',laserCMD)]:=' ';
    ServiceStr:='/C SERVICE LASER ';
    ShowCurrentLDT;
    writeln;
    writeln('Enter:');
    writeln;
    writeln('      M  specify minutes from now for laser
',CMDnoperiod);
    writeln;
    writeln('      T  specify time and date for laser
',CMDnoperiod);
    repeat key:=upcase(Readkey) until key in ['T','M'];
    ShowCurrentLDT;
    if key='T' then begin
       writeln;
```

```
        writeln('Enter date and time to turn the laser
',laserCMD);
        writeln;
        writeln('FORMAT: mo/dd/yr hr:mn xx');
        writeln;
        writeln('  where mo=month  dd=day     yr=year');
        writeln('        hr=hour   mn=minute  xx=AM or PM
');
        writeln;
        CursorOn;
        Readln(chronoStr);
        iend:=0;
        iend:= pos('M',chronoStr);
        if iend=0 then iend:=pos('m',chronoStr);
        if iend=0 then begin
           ClrScr;
           gotoxy(1,3);
           writeln('   Invalid Entry: please specify
             AM or PM. ');
           userwait;
           exit;
        end;
        if laserCMD='on. ' then ServiceStr := ServiceStr
+ 'ON DATE '
                           else ServiceStr := ServiceStr
+ 'OFF DATE ';
        for i:= 1 to pos(' ',chronoStr)
           do ServiceStr := ServiceStr+chronoStr[i];
        ServiceStr:= ServiceStr + 'TIME';
        for i:= pos(' ',chronoStr) to iend
           do ServiceStr := ServiceStr+chronoStr[i];
     end else begin
        writeln;
        writeln('Enter minutes from now when the laser
will turn ',laserCMD);
        writeln;
        writeln;
```

```
        writeln;
        CursorOn;
        Readln(chronoStr);
        val(chronoStr,nnn,valcode);
        if (valcode=0) and (nnn>0) and (nnn<999) then
begin
           Str(nnn:4,nnnnstr);
           for i:= 1 to 4 do if nnnnStr[i]=' ' then
nnnnStr[i]:='0';
           if laserCMD='on. ' then ServiceStr :=
ServiceStr + 'ON +'
                              else ServiceStr :=
ServiceStr + 'OFF +';
           ServiceStr := ServiceStr + nnnnStr;
        end;
     end;
     CursorOff;
     ServiceStr:=ServiceStr + '/i/t/P';  { priority
service }
     exec('C:\COMMAND.COM',ServiceStr);
     delay(100);
end;  { procedure IssueTimeCommand }

begin
TryToAddOn := false;
TryToAddOff:= false;
repeat
  window(1,1,80,25);
  ClrScr;
  centerln('POWER, SLA-1 Power Control Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');

  writeln;
  centerln('Laser On/Off Timer Menu');
  DrawBox(10,8,80-10,23,1);
```

```
window(13,9,80-15,22);
ClrScr;
{$I-}
assign(chronofile,'\Chrono.sys');
reset(chronofile);
fileexists := (IOResult = 0);
if not fileexists then begin
   gotoxy(1,6);
   writeln('*** ERROR : CHRONO.SYS file not found.');
   UserWait;
   exit;
end else begin
   ShowCurrentLDT;
   chronoStr:='';
   fileempty := true;
   endoffile := false;
   OnEntryExists := false;
   OffEntryExists := false;
   for j:=1 to 2 do begin
   if not endoffile then begin
   repeat
      Readln(chronofile,ChronoStr);
      endoffile:=EOF(chronofile);
   until endoffile or ((chronostr[1]='A') and
(chronostr[3]='L'));
   if (not endoffile or (chronostr[1]='A')) then
fileempty:=false;
   if (not fileempty) and (chronostr[1]='A') then
begin
      if chronoStr[4]='O' then begin
         write('The laser is set to turn on ');
         OnEntryExists:=true
      end else begin
         write('The laser is set to turn off ');
         OffEntryExists:=true
      end;
      for i:= 1 to 8 do begin
```

```
            AbsDate[i]:=chronoStr[i+5];
            AbsTime[i]:=chronoStr[i+14];
         end;
         for i:= 1 to 2 do begin
            OffsetDay[i]:=chronoStr[i+23];
            OffsetHour[i]:=chronoStr[i+27];
            OffsetMonth[i]:=chronoStr[i+30];
         end;
         OffsetDay[3]:=chronoStr[26];
         writeln('at ');
         if (OffsetDay<>'000') or (OffsetHour<>'00') or
(OffsetMonth<>'00')
         then begin
            write('      ');
            if OffsetMonth<>'00' then write(OffsetMonth,'
month(s) ');
            if OffsetHour <>'00' then write(OffsetHour,'
hour(s) ');
            if OffsetDay <>'000' then write(OffsetDay,'
day(s) ');
            writeln('from ');
         end;
         writeln('                ',AbsTime,' on
',DazeType[DayOfWeekZeller],', ',AbsDate,'. ');
         writeln;
      end; {repeat}
      end; {if not end of file}
      end; {for j:= 1 to 2 }
   end;
   close(chronofile);
   {$I+}
   if not TryToAddOn and not TryToAddOff
   and not OnEntryExists and not OffEntryExists then
begin
      writeln;
      writeln('      No entries found in the laser timer
queue.');
```

```
  end;
  if TryToAddOn and not OnEntryExists
  then writeln('Invalid ON entry ignored.');
  if TryToAddOff and not OffEntryExists
  then writeln('Invalid OFF entry ignored.');
  gotoxy(1,12);
  writeln('Select:
  1. Remove ON entry      3. Add ON entry ');
  writeln('
  2. Remove OFF entry     4. Add OFF entry');
  write  ('
  ( X to exit )                          ');
  repeat key:=upcase(readkey) until key in
['1'..'4','X'];
  if key='X' then exit;
  ClrScr;
  TryToAddOn:=false;
  TryToAddOff:=false;
  if key='1' then exec('C:\COMMAND.COM','/C SERVICE
LASER ON CANCEL/i/t');
  if key='2' then exec('C:\COMMAND.COM','/C SERVICE
LASER OFF CANCEL/i/t');
  if key='3' then if OnEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser ON entry already
exists.  ');
     UserWait;
  end else IssueTimeCommand('on. ');
  if key='4' then if OffEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser OFF entry already
exists. ');
     UserWait;
  end else IssueTimeCommand('off.');
until false;
end;
```

```
procedure HardReset;
begin
  port[PowerPort + 3] := $98;    { init 8255 chip }
  port[PowerPort + 3] := $98;
end;

procedure InitPowerPort;
begin
  PowerControl(1,Off);
  if port[PowerPort + 2] and _INIT_IN > 0 then HardReset

  else begin
    PowerControl(1,On);
    if port[PowerPort + 2] and _INIT_IN = 0 then
HardReset;
  end;
end;

procedure ReadParams;
begin
  assign(ParamFile,ParamFileName);
  {$I-}
  reset(ParamFile);
  IOerr := IOresult;
  if IOerr <> 0 then begin
    assign(ParamFile,HomeDirectory + ParamFileName);
    reset(ParamFile);
    IOerr := IOresult;
  end;
  {$I+}
  LaserPower := _Unknown;
  if IOerr = 0 then begin
    while not eof(ParamFile) do begin
      readln(ParamFile,ParamFileLine);
      ParamFileLine := upper(ParamFileLine);
      if (ParamFileLine <> '') and (ParamFileLine[1] <>
'!') then begin
```

```
        if copy(ParamFileLine,1,15) = 'LASERPOWERTYPE='
then begin
          temp := copy(ParamFileLine,16,3);
          if temp = 'LEV' then LaserPower := _Level
          else if temp = 'TOG' then LaserPower :=
_Toggle;
        end;
      end;
    end;
    close(ParamFile);
  end;
end;

begin
  CheckBreak := false;
  ShowStuff  := false;
  LaserOff   := false;
  ClrScr;
  InitPowerPort;
  IndentStr := many(' ',Indent);
  ReadParams;
  CursorOff;
  repeat
    ShowPowerMenu;
    GetSelection;
    gotoxy(1,22);
    ClrEol;
    case key of
      '1': exec('C:\COMMAND.COM','/C SERVICE LASER ON
           NOW/i/t'){TurnOnLaser};
      '2': TurnOnElevator;
      '3': OpenShutter;
      '4': exec('C:\COMMAND.COM','/C SERVICE LASER OFF
           NOW/i/t'){TurnOffLaser};
      '5': TurnOffElevator;
      '6': CloseShutter;
      '7': SetLaserOnOffTime;
```

```
            'X': Abort;
          end;
        until false;
      end.
```

```
{=========================================================
  3D Systems Stereolithography System Software
=========================================================
   MATERIAL.PAS
   SLA-1 Material Manager Software
---------------------------------------------------------

   History:
   12/01/87  Ver 2.30  first release
   12/14/87  Ver 2.40  handles line widths min and
                       max, computes MSA
   12/17/87  Ver 2.41  FN zero conditions handled
                       (divide, LOG)
    1/08/88  Ver 2.50  relabeled Calc MSA due to MSA
                       concept problems
    1/25/88  Ver 2.60  version update only
    3/09/88  Ver 2.80  added copyright notice
    4/09/88  Ver 3.00  production release
    5/18/88  Ver 3.10  upgrade to turbo pascal
                       version 4.0
    7/18/88  Ver 3.20  update VersionId
    8/11/88  Ver 3.21  change LH input menu to say
                       1 (mil) or greater
    8/11/88  Ver 3.21  add if condition to recognize
                       new file in material
    8/11/88  Ver 3.21  update VersionId
    9/06/88  Ver 3.22  option 3 message if editing an
                       existing file & error check.

---------------------------------------------------------}

{$I-,R-,S+,D+,T-,F-,V+,B-,N-,L+ }
{$M $4000,0,0}

uses DOS, CRT;

const
```

```
  VersionId = '3.22';
  ScanFlag = FALSE;
  PowerFile          = 'POWER.LST';
  LastMaterialFile   = 'MATERIAL.LST';

type
  str80 = string[80];
  array20type = array [1..20] of real;

var
  key  : char;
  hereY : byte;
  file7, file9           : text;
  MaterialFileStr, InputStr, Tstr   : str80;
  SP,LH,WMIN,WMAX              : array20type;
  FileExists, StopLoop : boolean;
  LHSlope, LHYint,LHRval,WMINSlope,WMINYint,WMINRval
:real;
  WMAXSlope,WMAXYint,WMAXRval : real;
  i, j, valcode, NumberofPairs : integer;
  MaterialLaserPower : real;
  XY,SUMX,SUMY,SUMY2,XSQUARED : real;
  Slope,Yint,Rval,Denom,T, WMINx, WMAXx     : real;

procedure ShowTop;
begin
  ClrScr;

writeln('-------------------------------------------');
 writeln;
  writeln('  MATERIAL, Material Manager');
  writeln('  Version ',VersionId );
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography
            System');
  writeln;
```

```
writeln('-----------------------------------------');
  writeln;
end;

procedure UserWait;
begin
   writeln;
   writeln('                          ... press any key to
continue ');
   repeat delay(100) until keypressed;
   key := upcase(ReadKey);
end;

function RealVal(line: string;var code1: Integer):Real;
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
      line := '-0' + line;
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code1);
   if code1 <> 0 then
val(copy(line,1,code1-1),num,code1); { was code2 }
   RealVal := num;
end;

function Log10(n:real) : real;
begin
   { from Schaum's mathmatical handbook }
   Log10 := 0.434294481903 * ln(n);
end;
```

```
function power10(number:real) : real;
begin
   { // log(10**x) = x // log A**x = x log A //
  A**x =  e**x log A where ** indicates "to the power
          of" }
    power10 := exp(number * ln(10));
end;

function GetLog(var DataPoint,Slope,Yint: real): real;
begin
  if DataPoint > 0 then GetLog := Slope *
    log10(DataPoint) + Yint
  else GetLog := 0
end;

procedure Regression(var RArray : array20type;
  var Slope,Yint,Rval: real);
begin
  XY       := 0;
  SUMX     := 0;
  SUMY     := 0;
  SUMY2    := 0;
  XSQUARED := 0;
  if NumberOfPairs<1 then begin
    Slope  := 0;
    Yint   := 0;
    Rval   := 0;
    exit;
  end;
  for i:= 1 to NumberOfPairs do begin
    XY := XY + SP[i] * RArray[i];
    SUMX := SUMX + SP[i];
    SUMY := SUMY + RArray[i];
    SUMY2 := SUMY2 + sqr(RArray[i]);
    XSQUARED := XSQUARED + sqr(SP[i]);
  end;
  Denom := XSQUARED - sqr(SUMX) / NumberOfPairs;
```

```
  if Denom < 1E-5 then Slope := 0
  else Slope := (XY - (SUMX*SUMY)/NumberOfPairs) /
Denom;
  Yint := SUMY/NumberOfPairs - Slope *
    SUMX/NumberOfPairs;
  Denom := SUMY2 - sqr(SUMY) / NumberOfPairs;
  if Denom < 1E-5 then Rval := 0
  else Rval := SQR(((Yint * SUMY + Slope * XY) -
sqr(SUMY)/NumberOfPairs) / Denom);
end;  { procedure Regression }

procedure LoadMaterial;
begin
  ShowTop;
  writeln('Available Material Files:');
  writeln;
  exec('c:\command.com','/c dir *.MAT/P');
  if DOSerror <> 0 then writeln('  - no material files
    in working directory -');
{  repeat     }
    writeln;
    writeln('   Input name of material data file to
     read:');
    readln(MaterialFileStr);
    if length(MaterialFileStr)=0 then exit;
{  until length(MaterialFileStr)<>0; }
  for i:= 1 to length(MaterialFileStr) do
MaterialFileStr[i]
                                          :=
upcase(MaterialFileStr[i]);
  if pos('.',MaterialFileStr)=0 then MaterialFileStr
    := MaterialFileStr + '.MAT';
  assign(file7,MaterialFileStr);
  reset(file7);
  if IOresult <> 0 then begin
    writeln('   Error: cannot open material file
',MaterialFileStr);
```

```
      delay(2000);
    end else begin
      readln(file7,MaterialLaserPower);
      readln(file7,NumberofPairs);
      for i:= 1 to NumberOfPairs do begin
        readln(file7,SP[i]);
        readln(file7,LH[i]);
        readln(file7,WMIN[i]);
        readln(file7,WMAX[i]);
        SP[i] := ln(SP[i]) / ln(10);
      end;
      close(file7);
      Regression(LH,LHSlope,LHYint,LHRval);
      Regression(WMIN,WMINSlope,WMINYint,WMINRval);
      Regression(WMAX,WMAXSlope,WMAXYint,WMAXRval);
    end;
    assign(file9,LastMaterialFile);
    reset(file9);
    if IOresult <> 0 then begin
      writeln('   Error: cannot open file
',LastMaterialFile);
      exit;
    end;
    rewrite(file9);
    writeln(file9,MaterialFileStr);
    close(file9);
  end;   { procedure LoadMaterial }

  procedure ViewMaterial;
  begin
    ShowTop;
    writeln;
    writeln('Data from Material File: ',MaterialFileStr);
    writeln;
    writeln;
    write  ('Material Test Laser Power Reading: ');
    writeln( MaterialLaserPower:3:2);
```

```
    writeln;
    writeln('Material Data Pairs: (',NumberOfPairs,'
total)');
    writeln;
    for i := 1 to NumberOfPairs do begin
      write(i:2);
      write('    SP = ',power10(SP[i]):4:2);
      gotoxy(20,whereY);
      write('LH = ',LH[i]:4:2);
      gotoxy(35,whereY);
      write('WMIN = ',WMIN[i]:4:2);
      gotoxy(52,whereY);
      writeln('WMAX = ',WMAX[i]:4:2);
    end;
    writeln;
    write('  LH:  Slope =');
    write( LHSlope:4:2);
    gotoxy(23,whereY);
    write('Y-Intercept =');
    write( LHYint:6:2);
    gotoxy(45,whereY);
    write('R-Value =');
    writeln( LHRval:6:2);
    write('WMIN:  Slope =');
    write( WMINSlope:6:2);
    gotoxy(23,whereY);
    write('Y-Intercept =');
    write( WMINYint:6:2);
    gotoxy(45,whereY);
    write('R-Value =');
    writeln( WMINRval:6:2);
    write('WMAX:  Slope =');
    write( WMAXSlope:6:2);
    gotoxy(23,whereY);
    write('Y-Intercept =');
    write( WMAXYint:6:2);
    gotoxy(45,whereY);
```

```
  write('R-Value =');
  writeln( WMAXRval:6:2);
  writeln;
  UserWait;
end;

procedure InputMaterial;
begin
  ShowTop;
  writeln;
  writeln('Input Material Data
----------------------------------------------------')
;
  repeat
     writeln;
     writeln('   Enter name of material data file to
receive output, ');
     write  ('   or press enter to quit: ');
     readln(MaterialFileStr);
     if length(MaterialFileStr) = 0 then exit;
     for i:= 1 to length(MaterialFileStr) do
MaterialFileStr[i]:=

upcase(MaterialFileStr[i]);
     if pos('.',MaterialFileStr)=0 then MaterialFileStr
:= MaterialFileStr + '.MAT';
     assign (file7,MaterialFileStr);
     reset(file7);
     if (IOResult = 0) and (MaterialFileStr <>'') then
begin
        fileexists := true;
        close(file7);
        writeln;
        writeln('   File name already exists, please try
again. ');
     end else fileexists := false;
  until FileExists = false;
```

```
  writeln;
  writeln('   Enter Material Test Laser Power Reading
(mW): ');
  readln(InputStr);
  MaterialLaserPower := Realval(InputStr,valcode);
     if (MaterialLaserPower <1) or (MaterialLaserPower
>1000) then begin
     writeln;
     writeln(' Error: Improper value');
     Userwait;
     exit;
     end;
  writeln('   How many step period/ line height data
pairs? ');
  readln(InputStr);
  val(InputStr,NumberOfPairs,valcode);
  writeln('   *** All line height measurements should be
1 (mil) or greater.');
  writeln;
  for i := 1 to NumberOfPairs do begin
     write('   Data pair #',i,': SP = ');
     readln(InputStr);
     val(InputStr,SP[i],valcode);
     gotoxy(30,whereY-1);
     write('LH = ');
     readln(InputStr);
     LH[i] := RealVal(InputStr,valcode);
     if (SP[i]=0) or (LH[i]=0) then begin
        writeln;
        writeln('   Error: SP or LH entry was zero.  ');

        UserWait;
        exit;
     end;
     if LH[i] < 1 then begin
        writeln;
```

```
            writeln('   Error: LH entry was less than 1. ');
            writeln('    All entries should be in mils.  ');
            UserWait;
            exit;
          end;
          SP[i] := ln(SP[i]) / ln(10);
          gotoxy(45,whereY-1);
          write('WMIN = ');
          readln(InputStr);
          WMIN[i] := RealVal(InputStr,valcode);
          gotoxy(60,whereY-1);
          write('WMAX = ');
          readln(inputStr);
          WMAX[i] := RealVal(InputStr,valcode);
        end;
        assign(file7,MaterialFileStr);
        reset(file7);
        if (IOResult <> 0) and (MaterialFileStr <>'') then
          rewrite(file7); { a new file is OK for saving input
      }
        writeln(file7,MaterialLaserPower:6:4);
        writeln(file7,NumberOfPairs);
        FOR I := 1 TO NumberOfPairs do begin
          writeln(file7,power10(SP[i]):6:14 );
          writeln(file7,LH[i]:6:14 );
          writeln(file7,WMIN[i]:6:14 );
          writeln(file7,WMAX[i]:6:14 );
        end;
        close(file7);
        Regression(LH,LHSlope,LHYint,LHRval);
        Regression(WMIN,WMINSlope,WMINYint,WMINRval);
        Regression(WMAX,WMAXSlope,WMAXYint,WMAXRval);
        assign(file9,LastMaterialFile);
        reset(file9);
        if IOresult <> 0 then begin
```

```
      writeln('   Error: cannot open file
',LastMaterialFile);
      exit;
    end;
    rewrite(file9);
    writeln(file9,MaterialFileStr);
    close(file9);
end;    { procedure InputMaterial }

procedure LayerInfo;
begin
    ShowTop;
    writeln;
    writeln('   LAYER THICKNESS INFORMATION');
    writeln;
    writeln;
    writeln('   Input desired layer thickness (mils): ');
    readln(Tstr);
    if length(Tstr)=0 then exit;
    T := Realval(Tstr,valcode);       { to simulate cure
depth }
    writeln;
    writeln;
    T:=T+6;
    WMINx := GetLog(T,WMINSlope,WMINYint);
    WMAXx := GetLog(T,WMAXSlope,WMAXYint);
{*  MSA = 180/3.1415926535*(2*T/(WMIN+WMAX))
 *  IF MSA<0 THEN MSA=0
 *  IF MSA>89.999 THEN MSA=90 **** this is wrong (kept
from basic code) }
    write('WMIN = ',WMINx:6:2);
    writeln('     WMAX = ',WMAXx:6:2);
 {* writeln( USING "   MSA = ###.##";MSA ** don't show
yet (kept from basic code)}
    writeln;
    UserWait;
end;
```

```
begin
  MaterialFileStr:='';
  key :='z';
  repeat
     ShowTop;
     writeln;
     if length(MaterialFileStr)>0
     then writeln('     Material File:
',MaterialFileStr)
     else writeln;
     writeln;
     writeln('     1. Load Material Data');
     writeln('     2. View Material Data');
     writeln('     3. Input New Material Data');
     writeln('     4. Layer Thickness Information');
     writeln('     X. Exit');
     writeln;
     write('     Option: ');
     repeat delay(100) until keypressed;
     key := upcase(Readkey);
     if key = '1' then LoadMaterial;
     if key = '2' then ViewMaterial;
     if key = '3' then InputMaterial;
     if key = '4' then LayerInfo;
   until key='X';
end.
^Z
```

```
{=========================================================
  3D Systems Stereolithography System Software
=========================================================
   XFER.PAS
   SLA-1 File Transfer Utility Software
 ---------------------------------------------------------

   History:
   11/12/87  Ver 2.30  first Beta software release
   12/02/87  Ver 2.31  reactivates tcp for Turbo Pascal
                       V4.0 Menu
   12/17/87  Ver 2.40  telnet/ftp host is
                       'SliceComputer' (not 'process')
    1/11/88  Ver 2.50  support of different Ethernet
                       types based on SLA1.SYS file
    1/20/88  Ver 2.51  upper case X for exit
    1/25/88  Ver 2.60  version update only
    3/09/88  Ver 2.80  added copyright notice
    4/04/88  Ver 3.00  change menus
    7/18/88  Ver 3.20  update VersionId
 ---------------------------------------------------------}


{$R-,S+,I+,D+,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,16384 }

uses Crt,Dos;

type
  EthernetSupport =
(_NoEthernet,_Excelan,_Micom,_Unknown);

var
  Ch: Char;
  Line,SingleBar: String;
  Ethernet: EthernetSupport;
  SystemFile,ErrorFile: Text;
```

```
 const
  VersionId = '3.20';
  ComSpec = 'C:\COMMAND.COM';
  SystemFileName = 'C:\SLA1.SYS';
  ErrorFileName = 'C:\SYSTEM.ERR';

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln(Text);
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'XFER: ',Text);
  close(ErrorFile);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): String;
var
  temp: String;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
```

```
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

procedure DetermineEthernetSupport;
begin
  Ethernet := _NoEthernet;
  assign(SystemFile,SystemFileName);
  {$I-}
  reset(SystemFile);
  {$I+}
  if IOresult=0 then begin
    while not eof(SystemFile) do begin
      readln(SystemFile,Line);
      Line := upper(Line);
      if copy(Line,1,9)='ETHERNET=' then begin
        case Line[10] of
          'N': Ethernet := _NoEthernet;
          'E': Ethernet := _Excelan;
          'M': Ethernet := _Micom;
          'U': Ethernet := _Unknown;
        end;
      end;
    end;
  end;
end;

begin
  DetermineEthernetSupport;
  SingleBar := many(#196,26);
  if Ethernet = _Excelan then Exec(ComSpec,'/C REX');
  repeat
    ClrScr;
```

```
    centerln('XFER, Ethernet File Transfer');
    centerln('Version '+VersionId);
    centerln('Copyright (C) 1988 by 3D Systems, Inc.');
    centerln('3D Systems Laser Stereolithography
      System');
    centerln(SingleBar);
    writeln;
    writeln;
    writeln;
    if Ethernet = _NoEthernet then begin
      SystemError('No Ethernet Communications Support');

      halt(1);
    end else begin
      case Ethernet of
        _Excelan: centerln('using Excelan Ethernet
support');
        _Micom  : centerln('using Micom Interlan
support');
        else      centerln('using unknown support');
      end;
      writeln;
      writeln;
      centerln('1. TELNET, Terminal Utility     ');
      centerln('2. FTP,    File Transfer Program');
      centerln('X. Exit Program                 ');
      writeln;
      writeln;
      gotoxy(32,whereY);
      write('Which action? ');
      Ch := upcase(ReadKey);
      if Ch in ['1','2','X'] then begin
        writeln;
        writeln;
        case upcase(Ch) of
          '1': begin
```

```
                    Exec(ComSpec,'/C TELNET
SliceComputer');
                    halt(0);
                  end;
            '2': begin
                    Exec(ComSpec,'/C FTP SliceComputer');
                    halt(0);
                  end;
            'X': halt(0);
          end;
        end;
      end;
    until 0=1;
end.
^Z
```

```
{=====================================================
   3D Systems Stereolithography System Software
=====================================================
   ZSTAGE.PAS
   SLA-1 Z-STAGE CONTROL PROGRAM
-----------------------------------------------------

   Recent History:
   11/12/87  Ver 2.30  first Beta software release
   11/15/87  Ver 2.31  fixed motor controller timing
                       eliminated need for NUM LOCK
    1/22/88  Ver 2.50  added motion status
                       can move specified distances
                       (inches) supports ZSTAGE.PRM file
    1/25/88  Ver 2.60  version update only
    2/24/88  Ver 2.61  Home Directory search for
                       ZSTAGE.PRM FIX displays source of
                       ZSTAGE.PRM file
    3/22/88  Ver 3.00  converted from Turbo Basic to
Turbo Pascal Version 4.0
    4/14/88  Ver 3.01  let any key stop the elevator
                       (dumb key)
    4/15/88  Ver 3.02  fix acceleration and velocity
                       send commands
    4/19/88  Ver 3.03  add code to process the limit
                       switches
    5/31/88  Ver 3.10  change elevator controller to
                       elevator mover
    7/18/88  Ver 3.20  update VersionId
    9/01/88  Ver 3.21  stop move given distance if
                       keypressed
-----------------------------------------------------}

uses CRT,DOS;
```

```
const
   VersionId     = '3.21';
   ParamFileName = 'ZSTAGE.PRM';
   HomeDirectory = 'c:\3DSYS\';
   Indent        = 19;            { use 22 for top lines
                                    of the display }
   Ctrlc         = ^C;
   Cr            = ^M;
   afterDecimal  =  4;

type
   str8  = string[ 8];
   str16 = string[16];
   str25 = string[25];
   str80 = string[80];

var
   ParamFileLine : String;
   CMD : str80;
   OpenParamFile, IndentStr : str25;
   InchDistanceStr, Accel, Veloc : str8;
   TimeoutValue, TC, ZAddress, i, m, n, ValCode, IOerr :
Integer;
   InchDistance, r : real;
   ParamFile : Text;
   BigDist,MicroSteps, j, MinimumZ : LongInt;
   ShowStuff, NegZStage, ShowParms, TimeOut, dumbkey  :
Boolean;
   regs: registers;
   key, Dir : char;

{$I combine.inc}
{$R-,S-,I-,F-,V-,B-}

function RealVal(line: string;var code: Integer):Real;
```

```
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
      line := '-0' + line;
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code);
   if code <> 0 then val(copy(line,1,code-1),num,code2);

   RealVal := num;
end;

procedure DefaultParms;
begin
   for i:=1 to 255 do ParamFileLine[i]:=' ';
   NegZStage :=   false;
   ShowStuff :=   false;
   ShowParms :=   false;
   ZAddress  :=     800;
   Accel     :=   '0.5';
   Veloc     :=   '  1';
   BigDist :=  2000000 ;
   MicroSteps := 100000;
   TimeoutValue := 2000;
end;

procedure ReadParms;
begin
   OpenParamFile := ParamFileName;
   assign(ParamFile,OpenParamFile);
   reset(ParamFile);
   IOerr := IOresult;
   if IOerr <> 0 then begin
```

```
      OpenParamFile := HomeDirectory+ParamFileName;
      assign(ParamFile,OpenParamFile);
      reset(ParamFile);
      IOerr := IOresult;
      if IOerr <> 0 then OpenParamFile := ' '
   end;
   if IOerr = 0 then begin
      i := 0;
      while not eof(ParamFile) do begin
         readln(ParamFile,ParamFileLine);
         ParamFileLine := upper(ParamFileLine);
         if ParamFileLine[1] = '"' then begin
            ParamFileLine[1] := ' ';
            m := pos('"',ParamFileLine);
            n := pos('/S',ParamFileLine);
            if (0<n) and (n<m) then ShowStuff := true;
            n := pos('/NEGZ',ParamFileLine);
            if (0<n) and (n<m) then NegZStage := true;
         end;
         if (ParamFileLine[1] <> ' ') and
(ParamFileLine[1] <> '!') and
            (ParamFileLine[1] <> '"') then begin
            if i < 6 then begin
               i := i+1;     { this allows blank lines
in the parameter file }
               val(ParamFileLine,j,valcode);
               if valcode>2 then
                  InchdistanceStr :=
copy(paramFileLine,1,valcode-1);
               val(InchDistanceStr,j,valcode);
               case i of
                  1 : ZAddress := j;
                  2 : Accel    := ParamFileLine;
                  3 : Veloc    := ParamFileLine;
                  4 : BigDist := j;
                  5 : MicroSteps := j;
                  6 : TimeoutValue := j;
```

```
                end;
              end;
            end;
          end;
        end;
      end;  { ReadParams }

    procedure ShowZStageMoverMenu;
      begin
        window(1,1,80,25);
        ClrScr;
        writeln(IndentStr,'     Elevator Mover          ');
       writeln(IndentStr,'  3D Systems Stereolithography
    System');
        writeln(IndentStr,'  Version '+VersionID+'
    3D Systems, Inc.');
        gotoxy(1,5);
        if TIMEOUT   then writeln(IndentStr,'
    ELEVATOR OFF ')
        else              writeln(IndentStr,'
    ELEVATOR STOP');

        DrawBox(Indent-1,7,80-Indent,21,1);
        window(Indent,9,79-Indent,21);
        writeln('   UP ARROW    move elevator up');
        writeln;
        writeln(' DOWN ARROW    move elevator down');
        writeln;
        writeln('  SPACE BAR    stop elevator motion');
        writeln;
        writeln('          D    move a specified
    distance');
        writeln;
        writeln('          T    toggle parameter
    display');
        writeln;
        writeln('     Q or X    exit to Utility menu');
```

```
      if ShowParms then begin
         window(Indent-6,22,80,25);
         if OpenParamFile = ' ' then
         writeln('      Not using any Parameter File:')
         else writeln('      Using '+OpenParamFile+'
Parameter File:');
         write('      Address=',Zaddress,'
Accel='+accel+'  Veloc='+Veloc);
         if NegZStage then writeln('  /NEGZ') else
writeln;
         writeln('Big Dist=',bigdist,'
Microsteps=',microsteps,'  Timeout
Value=',timeoutValue);
      end;
end;  { procedure ShowZStageMoverMenu }

procedure GetSelection;
begin
   key := 'Z';
   repeat
      HandleKeyboardRequest;
      key := upcase(key);
      if dumbkey then key:=' ';
      dumbkey:=false;
      if key in [#72,#80,' ','D','T','Q','X'] then exit;
{ up and down arrows }
   until false;
end;  { procedure GetSelection }

procedure SendElevCommand(Ch: Char; TimeOutValue:
integer; TimeOut: boolean);
var
   b, TC : integer;
begin
   TimeOut := false;
   TC      := 0;
```

```
   repeat                          { Read status byte to
determine if the input...}
      b := port[ZAddress+1];   { data buffer (IDB) is
ready for a character.  }
      TC := TC + 1;
   until ((b and $10)>0) or (TC > TimeOutValue); { Bit 4
true=1 means ready. }
   port[ZAddress] := ord(Ch); { Convert character to
ASCII, give to IDB.     }
   port[ZAddress+1] := $70;    { Reset status byte for
ready to send to motor.}
   repeat
      b:= port[ZAddress+1];    { Read status byte to
determine if the IDB...  }
      TC := TC + 1;            { read the instruction, if
not, wait until ... }
   until ((b and $10)=0) or (TC > TimeOutValue);
        { it does. }
   port[ZAddress+1] := $60;    { Reset status byte for
ready to send again.    }
   if TC > TimeOutValue then TimeOut := True;
end; { Send Elevator Command }

procedure ElevatorInterface(Cmd:
string;TimeOutValue:integer;TimeOut:boolean);
{ this is the gateway through which all elevator
commands must pass }
var
   ii : integer;
begin
 {  CMD := 'E MN A'+Accel+' V'+veloc+'+CMD; from SetAVD
procedure }
   if ShowStuff then begin
      writeln('  DEBUG Elevator Interface CMD='+Cmd);
   end;
   delay(1); { this is required for the program
configuration to work right }
```

```
   for ii := 1 to length(Cmd) do
        SendElevCommand(Cmd[ii],TimeOutValue,TimeOut);
end;  { Elevator Interface }

procedure GetIndexerResponse(var
Cmd:string;TimeOutValue:integer;TimeOut:boolean);
var
   b, response, TC1, TC2 : integer;
begin
   TC1  := 0;
   CMD := '';
   repeat
      if CMD<>'*' then begin
         CMD :='';
         ElevatorInterface('R ',TimeOutValue,TimeOut);
      end;
      repeat
      TC1 := TC1 + 1;
      TC2 := 0;
         repeat
            b := port[Zaddress+1];     { Read status
byte.                    }
            TC2 := TC2+1;
         until ((b and $08)<>0) or (TC2 > TimeOutValue);
        response := port[Zaddress];   { OK to read
response from controller. }
         port[ZAddress+1] := $E0;      { Set status byte
to message received. }
         repeat
            b := port[ZAddress+1];     { Wait for read
status to hit ready.   }
            TC2 := TC2 + 1;
         until ((b and $08)=0) or (TC2 > TimeOutValue);
         port[ZAddress+1] := $60;        { Reset status
byte.                 }
      until (chr(response) <> '') or (TC2 >
TimeOutValue);
```

```
      CMD :=CMD+chr(response);
      delay(5);
      until (CMD='*S') or (CMD='*R')
            or (keypressed) or (TC1 > TimeOutValue);
end; { Get elevator controller response }

procedure MoveGivenDistance;
var
   dir : str8;
begin
   window(Indent-8,22,79,25);
   writeln('     *** Key distance in inches (positive is
up)  ***           ');
   writeln('
                  ');
   writeln('
                  ');
   gotoxy(Indent,2);
   readln(inchdistancestr);                if
inchdistancestr[1] <> ' ' then begin
      inchdistance := RealVal(inchdistancestr,valcode);
      if inchdistance <> 0 then begin
         if not NegZStage then inchdistance :=
-inchdistance;
         j := trunc(inchdistance * microsteps);
         STR(j,inchdistancestr);
         CMD:='E MN A'+Accel+' V'+veloc+'
D'+inchdistancestr+' G ';
         ElevatorInterface(CMD,TimeOutValue,TimeOut);
         dir := 'DOWN  ';
         if ((j < 0) and (not NegZStage)) or ((j > 0)
and NegZSTage)
                                                then
dir := 'UP  ';
         window(1,5,80,6);
```

```
            if not TimeOut then writeln(IndentStr,'
  ELEVATOR '+dir);
            delay(40);
            GetIndexerResponse(CMD,TimeOutValue,TimeOut);
            ElevatorInterface('K ',TimeOutValue,TimeOut);
         end;
      end;
   end;    { procedure MoveGivenDistance }

procedure MoveTable(direction:string);
begin
   MinimumZ := BigDist;
   if direction = 'UP  ' then MinimumZ := - BigDist;
   CMD:='E MN A'+Accel+' V'+veloc+'
D'+RealStr(MinimumZ,0)+' G ';
   window(1,5,80,6);
   if not TimeOut then writeln(IndentStr+'
ELEVATOR '+direction);
   repeat
      ElevatorInterface(CMD,TimeOutValue,TimeOut);
      delay(40);
      GetIndexerResponse(CMD,TimeOutValue,TimeOut);
   until (KeyPressed) or (CMD='*S');
   GetIndexerResponse(CMD,TimeOutValue,TimeOut);
   if CMD<>'*S' then dumbkey := true;
 { key:=Readkey; }
   ElevatorInterface('K ',TimeOutValue,TimeOut);
end;  { procedure MoveTable }

begin { main program }
   ClrScr;
   window(1,1,80,25);
   IndentStr := many(' ',Indent);
   DefaultParms;
   ReadParms;
   CursorOff;
   dumbkey := false;
```

```
   TimeOut := false;
   InitZTable;

   repeat
      ShowZStageMoverMenu;
      GetSelection;
      case key of
       #72 :  if not NegZStage then MoveTable('UP  ')
else
                                        MoveTable('DOWN');
       #80 :  if not NegZStage then MoveTable('DOWN')
else
                                        MoveTable('UP  ');
        'D':  MoveGivenDistance;
        'T':  ShowParms := not(ShowParms);
        'Q':  Abort;
        'X':  Abort;
      end;
   until false;

   CursorOn;
   ClrScr;
end.  { main program }
```

```
{

   Bench
   Benchmark Unit
   3D Systems Stereolithography System

}

unit Bench;

{$R-,S-,I-,D-,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,0 }

interface

uses Dos;

const
  MAX_BENCH_ITEMS = 50;

type
  StringArrayPtr = ^StringArray;
  StringArray = array[0..MAX_BENCH_ITEMS] of String[16];

var
  Bench_Start_DOW: Word;
  Bench_Item_Total,Bench_Item_Stamp:
array[0..MAX_BENCH_ITEMS] of LongInt;

procedure BenchInit(BenchFileName: String);
procedure BenchStart(Item: Word);
procedure BenchStop(Item: Word);
procedure BenchEnd(BenchFileName:
String;Bench_Item_Title: StringArrayPtr);

implementation
```

```
procedure BenchInit(BenchFileName: String);
  var
    BenchFile: Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW: Word;
    i: Integer;
  begin
    for i:=1 to MAX_BENCH_ITEMS do begin
      Bench_Item_Total[i]:=0;
      Bench_Item_Stamp[i]:=0;
    end;
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BenchFileName);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    writeln(BenchFile,'Benchmark Started for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
    Bench_Start_DOW := DOW;
  end;

function GetS100Time: LongInt;
  var
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW: Word;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    if DOW<Bench_Start_DOW then Inc(DOW,7);
    GetS100Time := S100 + LongInt(Sec)*100 +
LongInt(Min)*6000 + Hr*360000 +
                   (DOW-Bench_Start_DOW)*8640000;
  end;

procedure BenchStart(Item: Word);
  begin
```

```
    if Bench_Item_Stamp[Item]=0 then
Bench_Item_Stamp[Item] := GetS100Time;
  end;

procedure BenchStop(Item: Word);
  begin
    if Bench_Item_Stamp[Item]>0 then begin
      Inc(Bench_Item_Total[Item],GetS100Time -
Bench_Item_Stamp[Item]);
      Bench_Item_Stamp[Item] := 0;
    end;
  end;

procedure BenchEnd(BenchFileName:
String;Bench_Item_Title: StringArrayPtr);
  var
    BenchFile: Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW: Word;
    i: Integer;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BenchFileName);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    for i:=1 to MAX_BENCH_ITEMS do
      if Bench_Item_Stamp[i]>0 then begin
        BenchStop(i);
        writeln(BenchFile,' Total
',Bench_Item_Title^[i],
                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec RUNNING');
      end else
        writeln(BenchFile,' Total
',Bench_Item_Title^[i],
                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec');
```

```
    writeln(BenchFile,'Benchmark Ended   for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
  end;

end.
^Z
```

```
{ NOTE: After Compulation, rename EXE file to BUILD.EXE
}

{
=========================================================
     GOBUILD, Sterolithography Part-Builder Front End
    for 3D Systems' Sterolithography System

=========================================================

 Recent History:
   4/20/88  Ver 3.10  Initial Release
   4/25/88  Ver 3.11  Correct command line pasing
}

{$R-,S-,I-,D-,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,0 } { No Heap Allocation }

Uses
  Dos;

  const
    DosName = 'C:\COMMAND.COM';
    ProgramName = '/C BUILDNC';
    ProgramCOPName = '/C BUILDC';

  var
    regs: Registers;
    COP: Boolean;
    CommandLine,ProgramStr : String;
    i: Integer;

  begin
    { Get Parameters }
    CommandLine:='';
    for i:=1 to ParamCount do CommandLine := CommandLine
+ ParamStr(i);
```

```
    { Make Equipment check call }
  Intr($11,regs);
  COP := (regs.AL and 2)<>0;

  { Select the correct file }
  if COP then ProgramStr := ProgramCOPName
  else ProgramStr := ProgramName;
  Exec(DosName,ProgramStr+' '+CommandLine);
  Halt(DosExitCode);
end.
```

```
{=========================================================
   3D Systems Stereolithography System Software
 =========================================================
  SERVICE.PAS
   SLA-1 Special Services Provider
---------------------------------------------------------


   Recent History:
   11/20/87  Ver 2.30  under development
   12/06/87  Ver 2.31  Error output shows SERVICE, not
                       MERGE
                       /I in command line causes all
                         errors to be ignored
                       /T in command line for
                         transparent operation
    1/25/88  Ver 2.60  version update only
    2/24/88  Ver 2.61  converted to Turbo Pascal
                       Version 4.0
                       added program title header
                       added invalid command error
    3/09/88  Ver 2.80  added copyright notice
    4/09/88  Ver 3.00  production release
    4/25/88  Ver 3.10  LASER service rename LCTRL
                       added LASER ON/OFF (physical
                       power) service updating of
                      \CHRONO.SYS file
    5/11/88  Ver 3.11  Add priority laser off processing
    7/18/88  Ver 3.20  Update VersionId
 ---------------------------------------------------------}


{$R-,S-,I-,D-,T-,F-,V-,B-,N-,L- }
{$M 16384,0,0 }

uses Dos,Crt;
```

```
const
  VersionId    = '3.20';
  MSDOS        = $21;    { interrupt vectors }
  LASERVECTOR = $64;
  ChronoFileName     = 'C:\CHRONO.SYS';
  ErrorFileName      = 'C:\SYSTEM.ERR';
  HomeDirectory      = 'C:\3DSYS\';
  PowerParamFileName = 'POWER.PRM';
  LaserBoardAddress  = $300;
  PowerPort          = $304;
  _INIT_OUT          = $01;   { outputs }
  _ZSTAGE_OUT        = $02;
  _SHUTTER_OUT       = $04;
  _LASER_OUT         = $08;
  _INIT_IN           = $10;   { inputs }
  _DOOR_IN           = $20;
  _SHUTTER_IN        = $40;
  _LASER_IN          = $80;

type
  str4  = string [4];
  str10 = string[10];
  str16 = string[16];
  str40 = string[40];
  LaserPowerType = (_Unknown,_Level,_Toggle);
  OnOffType = (On,Off);
  DateType = record
    year : Word;
    month: Word;
    day  : Word;
  end;
  TimeStampType = record
    year : Word;
    month: Byte;
    day  : Byte;
    hour : Byte;
    minu : Byte;
```

```
  end;
  YearType = (NotLeapYear,LeapYear);

const
  DayOfWeek: String[21] = 'SunMonTueWedThuFriSat';
  Zel: array[1..12] of Integer =
    (1,4,3,6,1,4,6,2,5,0,3,5);
  DaysInMonth: array[NotLeapYear..LeapYear,1..12] of
Word
    = ((31,28,31,30,31,30,31,31,30,31,30,31),
       (31,29,31,30,31,30,31,31,30,31,30,31));
  DayTable: array[NotLeapYear..LeapYear,1..13] of Word
    = ((0,31,59,90,120,151,181,212,243,273,304,334,365),
     (0,31,60,91,121,152,182,213,244,274,305,335,366));
  ClearTimeStamp: TimeStampType =
    (year:0;month:0;day:0;hour:0;minu:0);
type
  LaserStateType     = (_LaserOn,_LaserOff);
  ServiceRequestType = (_LCTRL_ON,_LCTRL_OFF,

_LASER_OFF_dto,_LASER_OFF_NOW,_LASER_OFF_CANCEL,
                        _LASER_ON_dto, _LASER_ON_NOW,
_LASER_ON_CANCEL);
  ChronoEntryType    = (_BW,_BP,_LO,_LF);
  UpdateActionType   = (_NONE,_ACTIVE,_SKIP,_SET);
  UpdateKeyStructure =
array[LaserStateType,ServiceRequestType,ChronoEntryType]
                        of UpdateActionType;
  ChronoEntryFormat = record
    FileIndex    : LongInt;
    SkipFlag     : Boolean;
    Op           : ChronoEntryType;
    WhenToExecute: LongInt;
    PeriodOffset : LongInt;
  end;

const
```

```
  NumChronoEntries = 4;
  ChronoUpdateKey: UpdateKeyStructure =
  (                                                    BW      BP
LO      LF      )
  { LASER ON,  LCTRL ON          }    (((_NONE,  _NONE,
_ACTIVE,_NONE  ),
  { LASER ON,  LCTRL OFF         }      (_NONE,  _NONE,
_SKIP,  _NONE  ),
  { LASER ON,  LASER OFF dto     }      (_NONE,  _NONE,
_NONE,  _SET   ),
  { LASER ON,  LASER OFF NOW     }      (_SKIP,  _SKIP,
_NONE,  _SKIP  ),
  { LASER ON,  LASER OFF CANCEL  }      (_NONE,  _NONE,
_NONE,  _SKIP  ),

  { LASER ON,  LASER ON dto      }      (_NONE,  _NONE,
_SET,   _NONE  ),
  { LASER ON,  LASER ON NOW      }      (_NONE,  _NONE,
_NONE,  _NONE  ),
  { LASER ON,  LASER ON CANCEL   }      (_NONE,  _NONE,
_SKIP,  _NONE  )),

  { LASER OFF, LCTRL ON          }     ((_NONE,  _NONE,
_ACTIVE,_NONE  ),
  { LASER OFF, LCTRL OFF         }      (_NONE,  _NONE,
_SKIP,  _NONE  ),

  { LASER OFF, LASER OFF dto     }      (_NONE,  _NONE,
_NONE,  _SET   ),
  { LASER OFF, LASER OFF NOW     }      (_NONE,  _NONE,
_NONE,  _NONE  ),
  { LASER OFF, LASER OFF CANCEL  }      (_NONE,  _NONE,
_NONE,  _SKIP  ),
```

```
    ( LASER OFF, LASER ON dto      )      (_NONE,  _NONE,
_SET,   _NONE  ),
    ( LASER OFF, LASER ON NOW      )
(_ACTIVE,_ACTIVE,_SKIP,  _NONE  ),
    ( LASER OFF, LASER ON CANCEL   )      (_NONE,  _NONE,
_SKIP,  _NONE  )));

var
  i,IOerr,code: Integer;
  PowerParamFile,ErrorFile: Text;
  regs: Registers;
  LaserVersion: str4;
  SingleBar,CmdLine,cmd,temp: String;

ShowErrors,NoText,PriorityLaserOff,LaserOffFutureRequest
: Boolean;
  CurrentFileEntryHasPriority : Boolean;
  PriorityAbsMinutes, NewLFAbsMinutes, LOFutureTime  :
LongInt;
  LaserPower: LaserPowerType;
  Mon,Day,Year,Hour,Minu,Sec,WeekDay: Integer;
  Line,CurrentDate,CurrentTime: String;
  SystemFile,ChronoFile: Text;
  MinTimeStamp: TimeStampType;
  CurrentMinutes: LongInt;
  UpdateFile: File of Char;
  ChronoArray: array[1..50] of ChronoEntryFormat;
  ChronoCount: Integer;
  AbsMins: LongInt;

function upper(text: string): string;
var
  i: Integer;
  temp: String;
begin
  temp := text;
```

```
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
```

```
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
var
  ofs,code2: Integer;
  num: Real;
begin
  ofs := 0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  RealVal := num;
end;

function CleanUpSpaces(Text: String): String;
var
  Temp: String;
  i: Integer;
  HaveSpace: Boolean;
begin
  i := 1;
  while (i < length(Text)) and (copy(Text,1,1)=' ') do
inc(i);
  HaveSpace := false;
  Temp := '';
  while i <= length(Text) do begin
    if Text[i]=' ' then HaveSpace := true
    else begin
      if HaveSpace then begin
        Temp := Temp + ' ' + Text[i];
        HaveSpace := false;
      end else Temp := Temp + Text[i];
```

```
      end;
      inc(i);
    end; {while i}
    CleanUpSpaces := Temp;
  end;
```

```
{-------------------------------------------------------
                Time-Date-Chronometry Routines
-------------------------------------------------------}

 {
   convert integer to a string
}
function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);

  IntStr := temp;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
```

```
    else LZero := temp;
end;

{
  Compute day-of-week using Harry's modified Zeller
Algorithm
}
function Zeller(month,day,year: Integer): Integer;
begin
  if month<3 then dec(year);
  Zeller := (Zel[month] + day + year + (year shr 2))
    mod 7;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;     { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;     { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;     { get hours }
```

```
            Hour := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $07;    { get day }
              Day := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $08;    { get month }
                Mon := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then begin
                  port[$70] := $09;    { get year }
                  Year := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then good :=
true;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
    Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
    Minu := (Minu shr 4) * 10 + (Minu and $0F);
    Hour := (Hour shr 4) * 10 + (Hour and $0F);
    Day  := (Day  shr 4) * 10 + (Day  and $0F);
    Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
    Year := (Year shr 4) * 10 + (Year and $0F);
    WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
```

```
procedure GetTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
  CurrentDate := tdow + ' ' + tmon + '/' + tday + '/' +
tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

 procedure RecordDate(onoff:integer);
var
   fileexists : boolean;
   fnlaserlog : text;
   wyear,wmonth,wday,wdayofweek : word;
   whour,wminute,wsecond,wsec100 : word;
begin
```

```
      {$I-}
      assign(fnlaserlog,'c:\HISTORY\laser.log');
      reset(fnlaserlog);
      fileexists := (IOResult = 0);
      if not fileexists then rewrite(fnlaserlog);
      append(fnlaserlog);
      GetTimeStamp;
      write(fnlaserlog,CurrentDate,' ',CurrentTime);
      Case onoff of
        1 : writeln(fnlaserlog,' Laser powered ON.  ');
        2 : writeln(fnlaserlog,' Laser powered OFF. ');
      end;
      close(fnlaserlog);
end; { record date }

function StrParam(var Str: String;ParamNum: Integer):
String;
var
  count,index,endi: Integer;
begin
  count := 1;
  index := 1;
  while (count < ParamNum) and (index <= length(Str)) do
begin
    if Str[index] = ' ' then inc(count);
    inc(index);
  end;
  endi := index;
  while (endi <= length(Str)) and (Str[endi] <> ' ') do
inc(endi);
  StrParam := copy(Str,index,endi-index);
end;

 function IntParse: Integer;
var
  num: Real;
  IntNum,code,code2: Integer;
```

```
begin
  if length(temp)=0 then IntNum := 0
  else begin
    while copy(temp,1,1) = ' ' do delete(temp,1,1);
    val(temp,num,code);
    if code<>0 then val(copy(temp,1,code-1),num,code2);
    IntNum := trunc(num);
    if code=0 then temp := ''
      else temp := copy(temp,code+1,99);
  end;
  IntParse := IntNum;
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
var
  s,e: Integer;
begin
  s := 1;
  while (Text[s] in ['"',' ']) and (s < length(Text))
    do inc(s);
  e := length(Text);
  while (Text[e] in ['"',' ']) and (e > s) do dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
   Tell user if not transparent
}
procedure Tell(Text: String);
begin
  if not NoText then writeln(Text);
end;
```

```
{
   read DOS command line
}
procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  writeln(Text);
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'SERVICE: ',Text);
  close(ErrorFile);
  end;

{
  Reset Computer System's Date and Time from Hardware
Clock
}
procedure SetSystemDateTime;
begin
  GetTime;
  SetDate(1900+Year,Mon,Day);
  SetTime(Hour,Minu,Sec,0);
end;
```

```
{
  Julian-Cummulative-Minute algorithms
}
function Julian(date: DateType): Word;
var
  YearLookup: YearType;
begin
  with date do begin
    if (year mod 4) = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
    Julian := 365 * (year-1900) +
DayTable[YearLookup,month] +
      day + ((year-1901) div 4) - 32142 {1/1/88};
  end;
end;

procedure UnJulian(JulNum: Word;var date: DateType);
var
  YearLookup: YearType;
  offset,j,lastj: LongInt;
begin
  with date do begin
    inc(JulNum,32142);   {1/1/88}
    year := 1900;
    j := 0;
    lastj := 0;
    while j <= JulNum do begin
      lastj := j;
      if year mod 4 = 0 then inc(j,366)
        else inc(j,365);
      inc(year);
    end;
    dec(year);   { back off a year }
    if year mod 4 = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
    offset := JulNum - lastj;
    month := 1;
```

```
    j := 0;
    lastj := 0;
    while j <= offset do begin
      lastj := j;
      j := DayTable[YearLookup,month];
      inc(month);
    end;
    dec(month,2);
    day := offset - lastj + 1;
  end;
end;

function AbsMinutes(TimeStamp: TimeStampType): LongInt;
var
  date: DateType;
begin
  with TimeStamp do begin
    date.year  := year;
    date.month := month;
    date.day   := day;
    AbsMinutes := LongInt(Julian(date)) * 1440 + (hour *
60 + minu);
  end;
end;

procedure GetCurrentMinutes;
begin
  GetTime;
  MinTimeStamp.year  := 1900 + Year;
  MinTimeStamp.month := Mon;
  MinTimeStamp.day   := Day;
  MinTimeStamp.hour  := Hour;
  MinTimeStamp.minu  := Minu;
  CurrentMinutes := AbsMinutes(MinTimeStamp);
end;
```

```
procedure AbsTimeStamp(Minutes: LongInt;var TimeStamp:
TimeStampType);
var
  date: DateType;
  DayMinutes: Word;
begin
  DayMinutes := Minutes mod 1440;
  UnJulian(Minutes div 1440,date);
  with TimeStamp do begin
    year  := date.year;
    month := date.month;
    day   := date.day;
    hour  := DayMinutes div 60;
    minu  := DayMinutes mod 60;
  end;
end;

procedure Hour12Format(var hour: Byte;var AmPmStr:
str4);
begin
  case hour of
    0    : begin
             hour := 12;
             AmPmStr := 'AM';
           end;
    1..11: AmPmStr := 'AM';
    12   : AmPmStr := 'PM';
    13..23: begin
              dec(hour,12);
              AmPmStr := 'PM';
            end;
  end;
end;

function AbsTimeStampStr(Minutes: LongInt): String;
var
  stamp: TimeStampType;
```

```
  AmPm: str4;
begin
  AbsTimeStamp(Minutes,stamp);
  with stamp do begin
    Hour12Format(hour,AmPm);
    AbsTimeStampStr := PadSp(month) + '/' + LZero(day) +
'/' + LZero(year-1900)
      + ' ' + PadSp(hour) + ':' + LZero(minu) + ' ' +
AmPm;
  end;
end;

procedure ReadChronoFile;
var
  Operation: str4;
  ChronoLine: String;
  TimeStamp: TimeStampType;
  DayOffset,HourOffset,MinuOffset: Integer;
  FileCharCount: LongInt;
begin
  ChronoCount := 0;
  LOFutureTime := 0;
  assign(ChronoFile,ChronoFileName);
  {$I-}
  reset(ChronoFile);
  {$I+}
  if IOresult<>0 then exit;
  FileCharCount := 0;
  while not eof(ChronoFile) do begin
    readln(ChronoFile,ChronoLine);
    if (ChronoLine <> '') and (ChronoLine[1] <> '!')
then begin
      inc(ChronoCount);
      ChronoLine := upper(ChronoLine);
      with ChronoArray[ChronoCount] do begin
        FileIndex := FileCharCount;
        if ChronoLine[1] = 'S' then SkipFlag := true
```

```
            else SkipFlag := false;
          Operation := copy(ChronoLine,3,2);
          if Operation = 'BW' then Op := _BW else
          if Operation = 'BP' then Op := _BP else
          if Operation = 'LO' then Op := _LO else
          if Operation = 'LF' then Op := _LF;
          temp := copy(ChronoLine,6,99);
          with TimeStamp do begin
            month := IntParse;
            day   := IntParse;
            year  := 1900 + IntParse;
            hour  := IntParse;
            minu  := IntParse;
            if copy(temp,1,2) = 'AM' then begin
              if hour = 12 then hour := 0;
            end else begin   ( PM )
              if hour < 12 then inc(hour,12);
            end;
            WhenToExecute := AbsMinutes(TimeStamp);
          end;
          DayOffset  := IntParse;
          HourOffset := IntParse;
          MinuOffset := IntParse;
          PeriodOffset := (LongInt(DayOffset) * 24 +
HourOffset) * 60
            + MinuOffset;
          if (ChronoLine[1]='A') and (operation= 'LO')
          then LOFutureTime := WhenToExecute;
          if LaserOffFutureRequest then begin
             if ChronoLine[(pos('!',ChronoLine)-1)] = 'P'
then begin
                CurrentFileEntryHasPriority := true;
                PriorityAbsMinutes := WhenToExecute;
             end else CurrentFileEntryHasPriority :=
false;
          end;
        end; {with ChronoArray}
```

```
    end; {ChronoLine}
    inc(FileCharCount,length(ChronoLine)+2);
  end; {while not eof}
  close(ChronoFile);
end;

procedure UpdateChronoFile;
var
  i,j: Integer;
  SkipChar, RoutineCharFlag, BlankChar : Char;
  ChronoLine,NewTimeStr: String;
  OldUpdateFilePos,UpdateFilePos: LongInt;
begin
  assign(UpdateFile,ChronoFileName);
  {$I-}
  reset(UpdateFile);
  {$I+}
  if IOresult<>0 then exit;
  for i:=1 to ChronoCount do with ChronoArray[i] do
begin
    Seek(UpdateFile,FileIndex);
    if SkipFlag then SkipChar := 'S'    { Skip   }
      else SkipChar := 'A';             { Active }
    write(UpdateFile,SkipChar);
    Seek(UpdateFile,FileIndex+5);
    NewTimeStr := AbsTimeStampStr(WhenToExecute);
    for j:=1 to length(NewTimeStr) do
      write(UpdateFile,NewTimeStr[j]);
    if LaserOffFutureRequest and (i=4) then begin
       BlankChar := ' ';
       RoutineCharFlag := 'P';
       Seek(UpdateFile,FileIndex+59);
       if PriorityLaserOff then
write(UpdateFile,RoutineCharFlag)
                          else
write(UpdateFile,BlankChar);
    end;
```

```
    end;
    close(UpdateFile);
end;
```

```
{-------------------------------------------------

                Laser Controller Routines
-------------------------------------------------}

{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
    if LaserOk then LaserPresent := true
```

```
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{-----------------------------------------------------}
function GetLaserState: LaserStateType;
begin
  if port[PowerPort + 2] and _LASER_IN = 0 then
GetLaserState := _LaserOn
    else GetLaserState := _LaserOff;
end;

procedure UpdateChrono(ServiceRequest:
ServiceRequestType);
var
  i,j: Integer;
  LaserState: LaserStateType;
  Action: UpdateActionType;
begin
  ReadChronoFile;
  LaserState := GetLaserState;
  for i:=0 to NumChronoEntries-1 do begin
    Action :=
ChronoUpdateKey[LaserState,ServiceRequest,ChronoEntryTyp
e(i)];
    if Action <> _NONE then begin
      j := 1;
      while (j <= ChronoCount) and (ChronoArray[j].Op <>
ChronoEntryType(i))
        do inc(j);
      if j <= ChronoCount then with ChronoArray[j] do
begin
```

```
          case Action of
            _ACTIVE: SkipFlag        := false;
            _SKIP  : SkipFlag        := true;
            _SET   : begin
                       WhenToExecute := AbsMins;
                       SkipFlag      := false;
                     end;
          end;
          if LaserOffFutureRequest then NewLFAbsMinutes :=
AbsMins;
        end;
      end; {Action <>}
    end; {for i}
    if LaserOffFutureRequest and (not PriorityLaserOff)
    and CurrentFileEntryHasPriority
    and (CurrentMinutes     < PriorityAbsMinutes)
    and (PriorityAbsMinutes < NewLFAbsMinutes   ) then
exit;
    UpdateChronoFile;
end;

{-------------------------------------------------}
procedure TurnLaserControllerOff;
var
    Er: Integer;
begin
    with regs do begin
      AX := $0100;
      Intr(LASERVECTOR,regs);
      Er := AL;
      if ShowErrors then begin
        if Er=0 then Tell('Laser Controller turned off.')
        else Tell('Error'+IntStr(Er)+' attempting to turn
laser off.');
      end;
    end;
    UpdateChrono(_LCTRL_OFF);
```

```
end;

procedure TurnLaserControllerOn;
var
  Er: Integer;
begin
  with regs do begin
    AX := $0000;
    BX := 1190;   { 1 ms step period }
    Intr(LASERVECTOR,regs);
    Er := AL;
    if ShowErrors then begin
      if Er=0 then Tell('Laser Controller turned on.')
      else Tell('Error'+IntStr(Er)+' attempting to turn
laser on.');
    end;
  end;
  UpdateChrono(_LCTRL_ON);
end;

procedure LaserControllerService;
begin
  if not LaserPresent then halt(1);
  if copy(cmd,7,2)='ON' then TurnLaserControllerOn
  else if copy(cmd,7,3)='OFF' then
TurnLaserControllerOff
  else Tell('Invalid LCTRL request.');
end;

{-----------------------------------------------------
                    Laser Power Routines
-----------------------------------------------------}
procedure PowerControl(bit: Integer;Op: OnOffType);
begin
  if Op = On then
    port[PowerPort + 2] := port[PowerPort + 2] or bit
  else
```

```
    port[PowerPort + 2] := port[PowerPort + 2] and
($FF-bit);
end;

procedure ToggleLaserPowerOn;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN = 0) or
(loop1 > 100000);
  PowerControl(_LASER_OUT,Off);
  if (loop1 > 100000) then
    Tell('Can''t detect laser toggle feedback.');
end;

procedure ToggleLaserPowerOff;
var
  loop1: LongInt;
begin
  PowerControl(_LASER_OUT,On);
  loop1 := 0;
  repeat
    inc(loop1);
  until (port[PowerPort + 2] and _LASER_IN > 0) or
(loop1 > 100000);
  PowerControl(_LASER_OUT,Off);
  if (loop1 > 100000) then
    Tell('Can''t detect laser toggle feedback.');
end;

procedure TurnOnLaser;
var
  loop: LongInt;
```

```
begin
  if LaserPower = _Toggle then begin
    if port[PowerPort + 2] and _LASER_IN > 0 then
ToggleLaserPowerOn;
  end else PowerControl(_LASER_OUT,On);
  recordDate(1);
end;

procedure OpenShutter;
begin
  PowerControl(_SHUTTER_OUT,On);
end;

function AllowLaserOffAfterWarning:boolean;
var
   timekeeper : integer;
   mykey : char;
begin
   ClrScr;
   for timekeeper:= 1 to 6 do writeln;
   writeln('     *****     FINAL WARNING BEFORE LASER
POWER IS SHUT OFF     *****');
   writeln;
   writeln('                press Y to commence final
shutdown...        ');
   writeln('                press any other key to halt
shutdown process ');
   writeln;
   writeln('     If no action is taken, the laser will
power off in 2 minutes.');
   timekeeper:= 0;
   mykey:='Y';
   repeat
      timekeeper := timekeeper +1;  { roughly 1 second }

      sound(1000);
      delay(500);
```

```
      nosound;
      delay(500);
   until Keypressed or (timeKeeper=120);
   if Keypressed then mykey := upcase(ReadKey);
   if myKey = 'Y' then AllowLaserOffAfterWarning := true

   else AllowLaserOffAfterWarning := false;
end;

procedure TurnOffLaser;
begin
  if LOFutureTime > 0 then begin
     GetCurrentMinutes;
     if (CurrentMinutes+240 > LOFutureTime) and
     ( LOFutureTime > CurrentMinutes ) then begin
        writeln('Laser power not shut off due to future
on time scheduled within the');
        writeln('next four hours. ');
        delay(400);
        openshutter;
        exit;
     end;
  end;
  if AllowLaserOffAfterWarning then begin
     if LaserPower = _Toggle then begin
       if port[PowerPort + 2] and _LASER_IN = 0 then
ToggleLaserPowerOff;
     end else PowerControl(_LASER_OUT,Off);
     recordDate(2);
  end else openshutter;
end;

procedure CloseShutter;
begin
  PowerControl(_SHUTTER_OUT,Off);
end;
```

```
procedure HardReset;
begin
  port[PowerPort + 3] := $98;    { init 8255 chip }
  port[PowerPort + 3] := $98;
end;

procedure InitPowerPort;
begin
  PowerControl(1,Off);
  if port[PowerPort + 2] and _INIT_IN > 0 then HardReset

  else begin
    PowerControl(1,On);
    if port[PowerPort + 2] and _INIT_IN = 0 then
HardReset;
  end;
end;

procedure ReadPowerParams;
var
  ParamFileLine: String;
begin
  assign(PowerParamFile,PowerParamFileName);
  {$I-}
  reset(PowerParamFile);
  IOerr := IOresult;
  if IOerr <> 0 then begin
    assign(PowerParamFile,HomeDirectory +
PowerParamFileName);
    reset(PowerParamFile);
    IOerr := IOresult;
  end;
  {$I+}
  LaserPower := _Unknown;
  if IOerr = 0 then begin
    while not eof(PowerParamFile) do begin
      readln(PowerParamFile,ParamFileLine);
```

```
      ParamFileLine := upper(ParamFileLine);
      if (ParamFileLine <> '') and (ParamFileLine[1] <>
'!') then begin
        if copy(ParamFileLine,1,15) = 'LASERPOWERTYPE='
then begin
          temp := copy(ParamFileLine,16,3);
          if temp = 'LEV' then LaserPower := _Level
          else if temp = 'TOG' then LaserPower :=
_Toggle;
        end;
      end;
    end;
    close(PowerParamFile);
  end;
end;

procedure HaveDate(var TimeStamp: TimeStampType;var
StrParamIndex: Integer);
begin
  inc(StrParamIndex);   { point to date entry }
  temp := StrParam(cmd,StrParamIndex);
  inc(StrParamIndex);  { point to 'TIME' text if present
}
  with TimeStamp do begin
    month := IntParse;
    day   := IntParse;
    year  := 1900 + IntParse;
  end;
end;

procedure HaveTime(var TimeStamp: TimeStampType;var
StrParamIndex: Integer);
var
  AmPmStr: String;
  TwelveHourFormat: Boolean;
begin
  TwelveHourFormat := false;
```

```
  inc(StrParamIndex);   { point to time entry }
  temp := StrParam(cmd,StrParamIndex);
  inc(StrParamIndex);   { point to AmPm text if present
}
  AmPmStr := upper(StrParam(cmd,StrParamIndex));
  if (copy(AmPmStr,1,2) = 'AM') or (copy(AmPmStr,1,2) =
'PM') then begin
    TwelveHourFormat := true;
    inc(StrParamIndex);
  end;
  with TimeStamp do begin
    hour := IntParse;
    minu := IntParse;
    if TwelveHourFormat then begin
      if AmPmStr[1] in ['A','P'] then begin
        case hour of
          1..11: if AmPmStr[1] = 'P' then inc(hour,12);
          12   : if AmPmStr[1] = 'A' then hour := 0;
        end;
      end;
    end; {length}
  end; {with TimeStamp}
end;

function GetFutureTimeStampStr: String;
var
  index: Integer;
  stamp: TimeStampType;
begin
  if copy(StrParam(cmd,3),1,1) = '+' then begin
    GetCurrentMinutes;
    AbsMins := CurrentMinutes +
LongVal(copy(StrParam(cmd,3),2,99),code);
  end else begin
    AbsMins := 0;
    stamp := ClearTimeStamp;
    index := 3;
```

```
    if StrParam(cmd,index) = 'DATE' then
HaveDate(stamp,index);
    if StrParam(cmd,index) = 'TIME' then
HaveTime(stamp,index);
    AbsMins := AbsMinutes(stamp);
  end;
  GetFutureTimeStampStr := AbsTimeStampStr(AbsMins);
end;

procedure TurnLaserPowerOnNow;
begin
  UpdateChrono(_LASER_ON_NOW);
  TurnOnLaser;
  OpenShutter;
  Tell('Laser power turned on.');
end;

procedure TurnLaserPowerOnCancel;
begin
  UpdateChrono(_LASER_ON_CANCEL);
  Tell('Laser power turn-on cancelled.');
end;

procedure TurnLaserPowerOnFuture;
var
  DateTimeStr: String;
begin
  DateTimeStr := GetFutureTimeStampStr;
  UpdateChrono(_LASER_ON_dto);
  Tell('Laser power to be turned on ' + DateTimeStr +
'.');
end;

procedure TurnLaserPowerOn;
begin
  { check laser off }
```

```
  if StrParam(cmd,3) = 'NOW' then TurnLaserPowerOnNow
    else if StrParam(cmd,3) = 'CANCEL' then
TurnLaserPowerOnCancel
      else if (StrParam(cmd,3) = 'DATE') or
(StrParam(cmd,3) = 'TIME')
        or (copy(StrParam(cmd,3),1,1) = '+') then
TurnLaserPowerOnFuture
          else Tell('Invalid LASER ON request.');
end;

procedure TurnLaserPowerOffNow;
begin
  UpdateChrono(_LASER_OFF_NOW);
  CloseShutter;
  TurnOffLaser;
  Tell('Laser power turned off.');
end;

procedure TurnLaserPowerOffCancel;
begin
  UpdateChrono(_LASER_OFF_CANCEL);
  Tell('Laser power turn-off cancelled.');
end;

procedure TurnLaserPowerOffFuture;
var
  DateTimeStr: String;
begin
  LaserOffFutureRequest := true;
  DateTimeStr := GetFutureTimeStampStr;
  UpdateChrono(_LASER_OFF_dto);
  Tell('Laser power to be turned off ' + DateTimeStr +
'.');
end;

procedure TurnLaserPowerOff;
begin
```

```
  { check laser on }
  if StrParam(cmd,3) = 'NOW' then TurnLaserPowerOffNow
    else if StrParam(cmd,3) = 'CANCEL' then
TurnLaserPowerOffCancel
      else if (StrParam(cmd,3) = 'DATE') or
(StrParam(cmd,3) = 'TIME')
        or (copy(StrParam(cmd,3),1,1) = '+') then
TurnLaserPowerOffFuture
          else Tell('Invalid LASER OFF request.');
end;

procedure LaserPowerService;
begin
  ReadPowerParams;
  if StrParam(cmd,2) = 'ON' then TurnLaserPowerOn
    else if StrParam(cmd,2) = 'OFF' then
TurnLaserPowerOff
      else Tell('Invalid LASER request.');
end;

{-----------------------------------------------------}

procedure ShowHeader;
begin
  SingleBar := many(chr(196),45);
  writeln;
  writeln(SingleBar);
  writeln('  SERVICE, SLA Resource Utility');
  writeln('  Version ',VersionId);
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography
System');
  writeln(SingleBar);
  writeln;
end;
```

```
{ - - - M A I N   P R O G R A M - - - }

begin
  ShowErrors := true;
  NoText     := false;
  PriorityLaserOff := false;
  LaserOffFutureRequest := false;
  ReadCommandLine(CmdLine);
  cmd := CleanUpSpaces(upper(CmdLine));
  i := pos('/I',cmd);
  if i>0 then begin
    ShowErrors := false;
    delete(cmd,i,2);
  end;
  i := pos('/T',cmd);
  if i>0 then begin
    NoText := true;
    delete(cmd,i,2);
  end;
  i := pos('/P',cmd);
  if i>0 then begin
    PriorityLaserOff := true;
    delete(cmd,i,2);
  end;
  if not NoText then ShowHeader;
  if length(cmd)>0 then begin
    if copy(cmd,1,5)='LCTRL' then LaserControllerService
    else if copy(cmd,1,5)='LASER' then LaserPowerService
    else Tell('Unrecognized command.');
  end;
  halt(0);
end.^Z
```

```
{=======================================================
   3D Systems Stereolithography System Software
=======================================================
   ZSTAGE.PAS
   SLA-1 Z-STAGE CONTROL PROGRAM
-------------------------------------------------------
   Recent History:
   11/12/87  Ver 2.30  first Beta software release
   11/15/87  Ver 2.31  fixed motor controller timing
                       eliminated need for NUM LOCK
    1/22/88  Ver 2.50  added motion status
                       can move specified distances
                       (inches) supports ZSTAGE.PRM file
    1/25/88  Ver 2.60  version update only
    2/24/88  Ver 2.61  Home Directory search for
                       ZSTAGE.PRM FIX displays source of
                       ZSTAGE.PRM file
    3/22/88  Ver 3.00  converted from Turbo Basic to
                       Turbo Pascal Version 4.0
    4/14/88  Ver 3.01  let any key stop the elevator
                       (dumb key)
    4/15/88  Ver 3.02  fix acceleration and velocity
                       send commands
    4/19/88  Ver 3.03  add code to process the limit
                       switches
    5/31/88  Ver 3.10  change elevator controller to
                       elevator mover
    7/18/88  Ver 3.20  update VersionId
    9/01/88  Ver 3.21  stop move given distance if
                       keypressed
-------------------------------------------------------}
uses CRT,DOS;

const
   VersionId     = '3.21';
   ParamFileName = 'ZSTAGE.PRM';
   HomeDirectory = 'c:\3DSYS\';
```

```
   Indent        = 19;            { use 22 for top lines
of the display }
   Ctrl C        = ^C;
   Cr            = ^M;
   afterDecimal  =  4;

type
   str8  = string[ 8];
   str16 = string[16];
   str25 = string[25];
   str80 = string[80];

var
   ParamFileLine : String;
   CMD : str80;
   OpenParamFile, IndentStr : str25;
   InchDistanceStr, Accel, Veloc : str8;
   TimeoutValue, TC, ZAddress, i, m, n, ValCode, IOerr :
Integer;
   InchDistance, r : real;
   ParamFile : Text;
   BigDist,MicroSteps, j, MinimumZ : LongInt;
   ShowStuff, NegZStage, ShowParms, TimeOut, dumbkey  :
Boolean;
   regs: registers;
   key, Dir : char;

{$I combine.inc}
{$R-,S-,I-,F-,V-,B-}

function RealVal(line: string;var code: Integer):Real;
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
```

```
      line := '-0' + line;
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code);
   if code <> 0 then val(copy(line,1,code-1),num,code2);

   RealVal := num;
end;

procedure DefaultParms;
begin
   for i:=1 to 255 do ParamFileLine[i]:=' ';
   NegZStage :=   false;
   ShowStuff :=   false;
   ShowParms :=   false;
   ZAddress  :=     800;
   Accel     :=   '0.5';
   Veloc     :=   '  1';
   BigDist :=  2000000 ;
   MicroSteps := 100000;
   TimeoutValue := 2000;
end;

procedure ReadParms;
begin
   OpenParamFile := ParamFileName;
   assign(ParamFile,OpenParamFile);
   reset(ParamFile);
   IOerr := IOresult;
   if IOerr <> 0 then begin
      OpenParamFile := HomeDirectory+ParamFileName;
      assign(ParamFile,OpenParamFile);
      reset(ParamFile);
      IOerr := IOresult;
      if IOerr <> 0 then OpenParamFile := ' '
   end;
```

```
   if IOerr = 0 then begin
      i := 0;
      while not eof(ParamFile) do begin
         readln(ParamFile,ParamFileLine);
         ParamFileLine := upper(ParamFileLine);
         if ParamFileLine[1] = '"' then begin
            ParamFileLine[1] := ' ';
            m := pos('"',ParamFileLine);
            n := pos('/S',ParamFileLine);
            if (0<n) and (n<m) then ShowStuff := true;
            n := pos('/NEGZ',ParamFileLine);
            if (0<n) and (n<m) then NegZStage := true;
         end;
         if (ParamFileLine[1] <> ' ') and
(ParamFileLine[1] <> '!') and
            (ParamFileLine[1] <> '"') then begin
            if i < 6 then begin
               i := i+1;     { this allows blank lines
in the parameter file }
               val(ParamFileLine,j,valcode);
               if valcode>2 then
                  InchdistanceStr :=
copy(paramFileLine,1,valcode-1);
               val(InchDistanceStr,j,valcode);
               case i of
                  1 : ZAddress := j;
                  2 : Accel    := ParamFileLine;
                  3 : Veloc    := ParamFileLine;
                  4 : BigDist := j;
                  5 : MicroSteps := j;
                  6 : TimeoutValue := j;
               end;
            end;
         end;
      end;
   end;
end;  { ReadParams }
```

```
procedure ShowZStageMoverMenu;
   begin
      window(1,1,80,25);
      ClrScr;
      writeln(IndentStr,'      Elevator Mover    ' );
      writeln(IndentStr,'  3D Systems Stereolithography
System');
      writeln(IndentStr,'  Version '+VersionID+'
3D Systems, Inc.');
      gotoxy(1,5);
      if TIMEOUT   then writeln(IndentStr,'
ELEVATOR OFF ')
      else              writeln(IndentStr,'
ELEVATOR STOP');
       DrawBox(Indent-1,7,80-Indent,21,1);
      window(Indent,9,79-Indent,21);
      writeln('   UP ARROW    move elevator up');
      writeln;
      writeln(' DOWN ARROW    move elevator down');
      writeln;
      writeln('  SPACE BAR    stop elevator motion');
      writeln;
      writeln('          D    move a specified
distance');
      writeln;
      writeln('          T    toggle parameter
display');
      writeln;
      writeln('     Q or X    exit to Utility menu');
      if ShowParms then begin
         window(Indent-6,22,80,25);
         if OpenParamFile = ' ' then
         writeln('      Not using any Parameter File:')
         else writeln('      Using '+OpenParamFile+'
Parameter File:');
         write('      Address=',Zaddress,'
Accel='+accel+'  Veloc='+Veloc);
```

```
        if NegZStage then writeln('  /NEGZ') else
writeln;
        writeln('Big Dist=',bigdist,'
Microsteps=',microsteps,'  Timeout
Value=',timeoutValue);
      end;
end;  { procedure ShowZStageMoverMenu }

procedure GetSelection;
begin
   key := 'Z';
   repeat
      HandleKeyboardRequest;
      key := upcase(key);
      if dumbkey then key:=' ';
      dumbkey:=false;
      if key in [#72,#80,' ','D','T','Q','X'] then exit;
{ up and down arrows }
   until false;
end;  { procedure GetSelection }

procedure SendElevCommand(Ch: Char; TimeOutValue:
integer; TimeOut: boolean);
var
   b, TC : integer;
begin
   TimeOut := false;
   TC      := 0;
   repeat                        { Read status byte to
determine if the input...}
      b := port[ZAddress+1];  { data buffer (IDB) is
ready for a character.  }
      TC := TC + 1;
   until ((b and $10)>0) or (TC > TimeOutValue); { Bit 4
true=1 means ready. }
   port[ZAddress] := ord(Ch); { Convert character to
ASCII, give to IDB.     }
```

```
   port[ZAddress+1] := $70;    { Reset status byte for
ready to send to motor.}
   repeat
      b:= port[ZAddress+1];    { Read status byte to
determine if the IDB...  }
      TC := TC + 1;            { read the instruction, if
not, wait until ... }
   until ((b and $10)=0) or (TC > TimeOutValue);
        { it does. }
   port[ZAddress+1] := $60;    { Reset status byte for
ready to send again.    }
   if TC > TimeOutValue then TimeOut := True;
end; { Send Elevator Command }

procedure ElevatorInterface(Cmd:
string;TimeOutValue:integer;TimeOut:boolean);
{ this is the gateway through which all elevator
commands must pass }
var
   ii : integer;
begin
 {  CMD := 'E MN A'+Accel+' V'+veloc+'+CMD; from SetAVD
procedure }
   if ShowStuff then begin
      writeln('  DEBUG Elevator Interface CMD='+Cmd);
   end;
   delay(1); { this is required for the program
configuration to work right }
   for ii := 1 to length(Cmd) do
        SendElevCommand(Cmd[ii],TimeOutValue,TimeOut);
end;  { Elevator Interface }

procedure GetIndexerResponse(var
Cmd:string;TimeOutValue:integer;TimeOut:boolean);
var
   b, response, TC1, TC2 : integer;
begin
```

```
   TC1  := 0;
   CMD := '';
   repeat
      if CMD<>'*' then begin
         CMD :='';
         ElevatorInterface('R ',TimeOutValue,TimeOut);
      end;
      repeat
      TC1 := TC1 + 1;
      TC2 := 0;
         repeat
            b := port[Zaddress+1];      { Read status
byte.                        }
            TC2 := TC2+1;
         until ((b and $08)<>0) or (TC2 > TimeOutValue);

         response := port[Zaddress];    { OK to read
response from controller. }
         port[ZAddress+1] := $E0;       { Set status byte
to message received. }
         repeat
            b := port[ZAddress+1];      { Wait for read
status to hit ready.    }
            TC2 := TC2 + 1;
         until ((b and $08)=0) or (TC2 > TimeOutValue);
         port[ZAddress+1] := $60;         { Reset status
byte.                    }
      until (chr(response) <> '') or (TC2 >
TimeOutValue);
      CMD :=CMD+chr(response);
      delay(5);
      until (CMD='*S') or (CMD='*R')
             or (keypressed) or (TC1 > TimeOutValue);
end; { Get elevator controller response }


procedure MoveGivenDistance;
```

```
var
   dir : str8;
begin
   window(Indent-8,22,79,25);
   writeln('     *** Key distance in inches (positive is
up)  ***           ');
   writeln('
                  ');
   writeln('
                  ');
   gotoxy(Indent,2);
   readln(inchdistancestr);                if
inchdistancestr[1] <> ' ' then begin
      inchdistance := RealVal(inchdistancestr,valcode);
      if inchdistance <> 0 then begin
         if not NegZStage then inchdistance :=
-inchdistance;
         j := trunc(inchdistance * microsteps);
         STR(j,inchdistancestr);
         CMD:='E MN A'+Accel+' V'+veloc+'
D'+inchdistancestr+' G ';
         ElevatorInterface(CMD,TimeOutValue,TimeOut);
         dir := 'DOWN  ';
         if ((j < 0) and (not NegZStage)) or ((j > 0)
and NegZSTage)
                                                then
dir := 'UP  ';
         window(1,5,80,6);
         if not TimeOut then writeln(IndentStr,'
  ELEVATOR '+dir);
         delay(40);
         GetIndexerResponse(CMD,TimeOutValue,TimeOut);
         ElevatorInterface('K ',TimeOutValue,TimeOut);
      end;
    end;
  end;   { procedure MoveGivenDistance }
```

```
procedure MoveTable(direction:string);
begin
   MinimumZ := BigDist;
   if direction = 'UP  ' then MinimumZ := - BigDist;
   CMD:='E MN A'+Accel+' V'+veloc+'
D'+RealStr(MinimumZ,0)+' G ';
   window(1,5,80,6);
   if not TimeOut then writeln(IndentStr+'
ELEVATOR '+direction);
   repeat
      ElevatorInterface(CMD,TimeOutValue,TimeOut);
      delay(40);
      GetIndexerResponse(CMD,TimeOutValue,TimeOut);
   until (KeyPressed) or (CMD='*S');
   GetIndexerResponse(CMD,TimeOutValue,TimeOut);
   if CMD<>'*S' then dumbkey := true;
 { key:=Readkey; }
   ElevatorInterface('K ',TimeOutValue,TimeOut);
end;  { procedure MoveTable }

begin { main program }
   ClrScr;
   window(1,1,80,25);
   IndentStr := many(' ',Indent);
   DefaultParms;
   ReadParms;
   CursorOff;
   dumbkey := false;
   TimeOut := false;
   InitZTable;

   repeat
      ShowZStageMoverMenu;
      GetSelection;
      case key of
       #72 :  if not NegZStage then MoveTable('UP  ')
else
```

```
                                        MoveTable('DOWN');
          #80 :  if not NegZStage then MoveTable('DOWN')
else
                                        MoveTable('UP  ');
           'D':  MoveGivenDistance;
           'T':  ShowParms := not(ShowParms);
           'Q':  Abort;
           'X':  Abort;
         end;
    until false;

    CursorOn;
    ClrScr;
end.  { main program }
```

```
{=======================================================
   3D Systems Stereolithography System Software
=======================================================
    MATERIAL.PAS
   SLA-1 Material Manager Software
-------------------------------------------------------

   History:
   12/01/87  Ver 2.30  first release
   12/14/87  Ver 2.40  handles line widths min and max,
                       computes MSA
   12/17/87  Ver 2.41  FN zero conditions handled
                       (divide, LOG)
    1/08/88  Ver 2.50  relabeled Calc MSA due to MSA
                       concept problems
    1/25/88  Ver 2.60  version update only
    3/09/88  Ver 2.80  added copyright notice
    4/09/88  Ver 3.00  production release
    5/18/88  Ver 3.10  upgrade to turbo pascal
                       version 4.0
    7/18/88  Ver 3.20  update VersionId
    8/11/88  Ver 3.21  change LH input menu to say
                       1 (mil) or greater
    8/11/88  Ver 3.21  add if condition to recognize new
                       file in material
    8/11/88  Ver 3.21  update VersionId
    9/06/88  Ver 3.22  option 3 message if editing an
                       existing file & error check.
 -------------------------------------------------------}
{$I-,R-,S+,D+,T-,F-,V+,B-,N-,L+ }
{$M $4000,0,0}

uses DOS, CRT;

const
  VersionId = '3.22';
  ScanFlag = FALSE;
```

```
  PowerFile          = 'POWER.LST';
  LastMaterialFile   = 'MATERIAL.LST';

type
  str80 = string[80];
  array20type = array [1..20] of real;

var
  key  : char;
  hereY : byte;
  file7, file9          : text;
  MaterialFileStr, InputStr, Tstr   : str80;
  SP,LH,WMIN,WMAX              : array20type;
  FileExists, StopLoop : boolean;
  LHSlope, LHYint,LHRval,WMINSlope,WMINYint,WMINRval
:real;
  WMAXSlope,WMAXYint,WMAXRval : real;
  i, j, valcode, NumberofPairs : integer;
  MaterialLaserPower : real;
  XY,SUMX,SUMY,SUMY2,XSQUARED : real;
  Slope,Yint,Rval,Denom,T, WMINx, WMAXx     : real;

procedure ShowTop;
begin
  ClrScr;

writeln('-----------------------------------------------
----------------');
  writeln;
  writeln('  MATERIAL, Material Manager');
  writeln('  Version ',VersionId );
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
  writeln('  3D Systems Laser Stereolithography
System');
  writeln;
```

```
writeln('-----------------------------------------------
-----------------');
  writeln;
end;

procedure UserWait;
begin
   writeln;
   writeln('                          ... press any key to
continue ');
   repeat delay(100) until keypressed;
   key := upcase(ReadKey);
end;

function RealVal(line: string;var code1: Integer):Real;
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
      line := '-0' + line;
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code1);
   if code1 <> 0 then
val(copy(line,1,code1-1),num,code1); { was code2 }
   RealVal := num;
end;

function Log10(n:real) : real;
begin
   { from Schaum's mathmatical handbook }
   Log10 := 0.434294481903 * ln(n);
end;
```

```
function power10(number:real) : real;
begin
   { // log(10**x) = x // log A**x = x log A // A**x =
e**x log A
        where ** indicates "to the power of" }
    power10 := exp(number * ln(10));
end;

function GetLog(var DataPoint,Slope,Yint: real): real;
begin
  if DataPoint > 0 then GetLog := Slope *
log10(DataPoint) + Yint
  else GetLog := 0
end;

procedure Regression(var RArray : array20type; var
Slope,Yint,Rval: real);
begin
  XY       := 0;
  SUMX     := 0;
  SUMY     := 0;
  SUMY2    := 0;
  XSQUARED := 0;
  if NumberOfPairs<1 then begin
    Slope  := 0;
    Yint   := 0;
    Rval   := 0;
    exit;
  end;
  for i:= 1 to NumberOfPairs do begin
    XY := XY + SP[i] * RArray[i];
    SUMX := SUMX + SP[i];
    SUMY := SUMY + RArray[i];
    SUMY2 := SUMY2 + sqr(RArray[i]);
    XSQUARED := XSQUARED + sqr(SP[i]);
  end;
```

```
  Denom := XSQUARED - sqr(SUMX) / NumberOfPairs;
  if Denom < 1E-5 then Slope := 0
  else Slope := (XY - (SUMX*SUMY)/NumberOfPairs) /
Denom;
  Yint := SUMY/NumberOfPairs - Slope *
SUMX/NumberOfPairs;
  Denom := SUMY2 - sqr(SUMY) / NumberOfPairs;
  if Denom < 1E-5 then Rval := 0
  else Rval := SQR(((Yint * SUMY + Slope * XY) -
sqr(SUMY)/NumberOfPairs) / Denom);
end;  { procedure Regression }

procedure LoadMaterial;
begin
  ShowTop;
  writeln('Available Material Files:');
  writeln;
  exec('c:\command.com','/c dir *.MAT/P');
  if DOSerror <> 0 then writeln('  - no material files
in working directory -');
{  repeat     }
    writeln;
    writeln('   Input name of material data file to
read:');
    readln(MaterialFileStr);
    if length(MaterialFileStr)=0 then exit;
{  until length(MaterialFileStr)<>0; }
  for i:= 1 to length(MaterialFileStr) do
MaterialFileStr[i]
                                          :=
upcase(MaterialFileStr[i]);
  if pos('.',MaterialFileStr)=0 then MaterialFileStr :=
MaterialFileStr + '.MAT';
  assign(file7,MaterialFileStr);
  reset(file7);
  if IOresult <> 0 then begin
```

```
      writeln('   Error: cannot open material file
',MaterialFileStr);
      delay(2000);
    end else begin
      readln(file7,MaterialLaserPower);
      readln(file7,NumberofPairs);
      for i:= 1 to NumberOfPairs do begin
        readln(file7,SP[i]);
        readln(file7,LH[i]);
        readln(file7,WMIN[i]);
        readln(file7,WMAX[i]);
        SP[i] := ln(SP[i]) / ln(10);
      end;
      close(file7);
      Regression(LH,LHSlope,LHYint,LHRval);
      Regression(WMIN,WMINSlope,WMINYint,WMINRval);
      Regression(WMAX,WMAXSlope,WMAXYint,WMAXRval);
    end;
    assign(file9,LastMaterialFile);
    reset(file9);
    if IOresult <> 0 then begin
      writeln('   Error: cannot open file
',LastMaterialFile);
      exit;
    end;
    rewrite(file9);
    writeln(file9,MaterialFileStr);
    close(file9);
  end;   { procedure LoadMaterial }

  procedure ViewMaterial;
  begin
    ShowTop;
    writeln;
    writeln('Data from Material File: ',MaterialFileStr);
    writeln;
    writeln;
```

```
  write  ('Material Test Laser Power Reading: ');
  writeln( MaterialLaserPower:3:2);
  writeln;
  writeln('Material Data Pairs: (',NumberOfPairs,'
total)');
  writeln;
  for i := 1 to NumberOfPairs do begin
    write(i:2);
    write('    SP = ',power10(SP[i]):4:2);
    gotoxy(20,whereY);
    write('LH = ',LH[i]:4:2);
    gotoxy(35,whereY);
    write('WMIN = ',WMIN[i]:4:2);
    gotoxy(52,whereY);
    writeln('WMAX = ',WMAX[i]:4:2);
  end;
  writeln;
  write('  LH:  Slope =');
  write( LHSlope:4:2);
  gotoxy(23,whereY);
  write('Y-Intercept =');
  write( LHYint:6:2);
  gotoxy(45,whereY);
  write('R-Value =');
  writeln( LHRval:6:2);
  write('WMIN:  Slope =');
  write( WMINSlope:6:2);
  gotoxy(23,whereY);
  write('Y-Intercept =');
  write( WMINYint:6:2);
  gotoxy(45,whereY);
  write('R-Value =');
  writeln( WMINRval:6:2);
  write('WMAX:  Slope =');
  write( WMAXSlope:6:2);
  gotoxy(23,whereY);
  write('Y-Intercept =');
```

```
  write( WMAXYint:6:2);
  gotoxy(45,whereY);
  write('R-Value =');
  writeln( WMAXRval:6:2);
  writeln;
  UserWait;
end;

procedure InputMaterial;
begin
  ShowTop;
  writeln;
  writeln('Input Material Data
----------------------------------------------------')
;
  repeat
     writeln;
     writeln('   Enter name of material data file to
receive output, ');
     write  ('   or press enter to quit: ');
     readln(MaterialFileStr);
     if length(MaterialFileStr) = 0 then exit;
     for i:= 1 to length(MaterialFileStr) do
MaterialFileStr[i]:=

upcase(MaterialFileStr[i]);
     if pos('.',MaterialFileStr)=0 then MaterialFileStr
:= MaterialFileStr + '.MAT';
     assign (file7,MaterialFileStr);
     reset(file7);
     if (IOResult = 0) and (MaterialFileStr <>'') then
begin
        fileexists := true;
        close(file7);
        writeln;
        writeln('   File name already exists, please try
again. ');
```

```
      end else fileexists := false;
    until FileExists = false;
    writeln;
    writeln('   Enter Material Test Laser Power Reading
(mW): ');
    readln(InputStr);
    MaterialLaserPower := Realval(InputStr,valcode);
      if (MaterialLaserPower <1) or (MaterialLaserPower
>1000) then begin
      writeln;
      writeln(' Error: Improper value');
      Userwait;
      exit;
      end;
    writeln('   How many step period/ line height data
pairs? ');
    readln(InputStr);
    val(InputStr,NumberOfPairs,valcode);
    writeln('   *** All line height measurements should be
1 (mil) or greater.');
    writeln;
    for i := 1 to NumberOfPairs do begin
      write('   Data pair #',i,': SP = ');
      readln(InputStr);
      val(InputStr,SP[i],valcode);
      gotoxy(30,whereY-1);
      write('LH = ');
      readln(InputStr);
      LH[i] := RealVal(InputStr,valcode);
      if (SP[i]=0) or (LH[i]=0) then begin
        writeln;
        writeln('   Error: SP or LH entry was zero.  ');

        UserWait;
        exit;
      end;
      if LH[i] < 1 then begin
```

```
            writeln;
            writeln('   Error: LH entry was less than 1. ');

            writeln('    All entries should be in mils.  ');

            UserWait;
            exit;
          end;
          SP[i] := ln(SP[i]) / ln(10);
          gotoxy(45,whereY-1);
          write('WMIN = ');
          readln(InputStr);
          WMIN[i] := RealVal(InputStr,valcode);
          gotoxy(60,whereY-1);
          write('WMAX = ');
          readln(inputStr);
          WMAX[i] := RealVal(InputStr,valcode);
        end;
        assign(file7,MaterialFileStr);
        reset(file7);
        if (IOResult <> 0) and (MaterialFileStr <>'') then
          rewrite(file7); { a new file is OK for saving input
      }
        writeln(file7,MaterialLaserPower:6:4);
        writeln(file7,NumberOfPairs);
        FOR I := 1 TO NumberOfPairs do begin
          writeln(file7,power10(SP[i]):6:14 );
          writeln(file7,LH[i]:6:14 );
          writeln(file7,WMIN[i]:6:14 );
          writeln(file7,WMAX[i]:6:14 );
        end;
        close(file7);
        Regression(LH,LHSlope,LHYint,LHRval);
        Regression(WMIN,WMINSlope,WMINYint,WMINRval);
        Regression(WMAX,WMAXSlope,WMAXYint,WMAXRval);
        assign(file9,LastMaterialFile);
        reset(file9);
```

```
  if IOresult <> 0 then begin
     writeln('   Error: cannot open file
',LastMaterialFile);
     exit;
  end;
  rewrite(file9);
  writeln(file9,MaterialFileStr);
  close(file9);
end;   { procedure InputMaterial }

procedure LayerInfo;
begin
  ShowTop;
  writeln;
  writeln('   LAYER THICKNESS INFORMATION');
  writeln;
  writeln;
  writeln('   Input desired layer thickness (mils): ');
  readln(Tstr);
  if length(Tstr)=0 then exit;
  T := Realval(Tstr,valcode);      { to simulate cure
depth }
  writeln;
  writeln;
  T:=T+6;
  WMINx := GetLog(T,WMINSlope,WMINYint);
  WMAXx := GetLog(T,WMAXSlope,WMAXYint);
{*  MSA = 180/3.1415926535*(2*T/(WMIN+WMAX))
 *  IF MSA<0 THEN MSA=0
 *  IF MSA>89.999 THEN MSA=90 **** this is wrong (kept
from basic code) }
  write('WMIN = ',WMINx:6:2);
  writeln('     WMAX = ',WMAXx:6:2);
 {* writeln( USING "   MSA = ###.##";MSA ** don't show
yet (kept from basic code)}
  writeln;
  UserWait;
```

```
end;

begin
  MaterialFileStr:='';
  key :='z';
  repeat
     ShowTop;
     writeln;
     if length(MaterialFileStr)>0
     then writeln('     Material File:
',MaterialFileStr)
     else writeln;
     writeln;
     writeln('     1. Load Material Data');
     writeln('     2. View Material Data');
     writeln('     3. Input New Material Data');
     writeln('     4. Layer Thickness Information');
     writeln('     X. Exit');
     writeln;
     write('     Option: ');
     repeat delay(100) until keypressed;
     key := upcase(Readkey);
     if key = '1' then LoadMaterial;
     if key = '2' then ViewMaterial;
     if key = '3' then InputMaterial;
     if key = '4' then LayerInfo;
   until key='X';
end.
^Z
```

```
(  To compile: use MKBUILD Batch File  )
(*
============================================================
   3D Systems Stereolithography System Software
============================================================
   BUILD.PAS
   SLA-1 Stereolithography Part-Builder
------------------------------------------------------------
  BUILD-specific Commands:
     ZW secs       -  set Z-axis Wait time in seconds
     ZD distance   -  set Z-axis dip Depth in millimeters
     ZV velocity   -  set Z-axis Velocity parameter
     ZA accel      -  set Z-axis Acceleration parameter
------------------------------------------------------------
    Recent History:
    9/ 1/87  Ver 2.04  Version 2 released
    9  2/87  Ver 2.06  variable dip logic corrected
    9/ 3/87  Ver 2.07  compiled by TBM (Modified Turbo
                       Basic)
                       double parse of options
                       EX & CL before double EC to
                       suppress glitch
    9/ 4/87  Ver 2.08  added SV, SI critical areas
    9/17/87  Ver 2.10  PRM numbers less than 0.1
                       supported
                       selectable range of Z layers with
                       /START, /STOP options
                       keyboard pause / continue / skip
                       to layer control
    9/22/87  Ver 2.11  fixed incorrect day of week read
    9/23/87  Ver 2.12  new SUPER commands ZW, ZD, ZV,
                       and ZA structural changes to
                       SUPER.PRM
    9/25/87  Ver 2.20  new SLICE 19 block format
                       supported
    9/28/87  Ver 2.21  /NEGZ Z-stage is negative flag
                       default option line in SUPER.PRM
```

		added NFUB support
		added lower-case SUPER command support
9/30/87	Ver 2.25	no execute for empty blocks (primarily for L block)
10/20/87	Ver 2.26	/DEMO out-of-vat-delay option for AutoFact '87
10/26/87	Ver 2.27	*.P files now *.L file error file support
11/03/87	Ver 2.28	special demo screen read from file SUPER.DMO
11/04/87	Ver 2.29	demo action descriptions with good timing
11/12/87	Ver 2.30	first Beta software release
11/15/87	Ver 2.31	added directory of *.L files
11/20/87	Ver 2.32	multiple part-making capability added
12/04/87	Ver 2.33	slice20 FUB FDB blocks Menu System forced pause at end of part-making run BLOCK.ID file supported
12/16/87	Ver 2.40	SUPER converted from Turbo Basic to Turbo Pascal faster file transveral, laser buffer filling no special demo screen support
12/18/87	Ver 2.41	fixed Z-Table dip routine (inf board output)
12/21/87	Ver 2.50	graphics support added start-up and part-making information displays laser vectors shown on screen
12/23/87	Ver 2.51	Block File read speed up skipping to new layers sped up 150 times
12/28/87	Ver 2.52	skip to new layer fixed Layer Control File command

		processing fixed
		added ability to flip between the two graphic screens
1/06/88	Ver 2.53	break abort allowed during skip action
		negative offsets shown as such
		wait action F1/F2 support added
		graphics display of error messages
		axes orientation arrows for view window
		better file directory display
1/22/88	Ver 2.54	control of min/max vat viewport coords via SUPER.PRM
		ShowFiles based on *.V instead of *.L
		case no longer important on SUPER.PRM option line
		better non-existent input files recovery
1/25/88	Ver 2.60	Graphics Supervisor no longer classified as Experimental
		support of *.R Range Control Files (as well as *.L)
		correct layer count derived from Range Control File
2/06/88	-	/DEMO option made functional
		rivet processing made functional
3/08/88	Ver 2.80	added copyright notice
		installed gain and offset drift correction
		implemented binary STEREO support
4/04/88	Ver 3.00	change messages and menus
4/08/88	Ver 3.01	added binary Vector File support
		removed Critical Area feature
		SUPER renamed to BUILD
4/11/88	Ver 3.02	fixed EofVectorFile routine

```
4/13/88   Ver 3.03   fixed skipped-to layer block
                     command default problem
4/15/88   Ver 3.04   added layer number to profiling
                     error messages
                     default part name input from last
                     part PREPARED
4/18/88   Ver 3.05   Change name of graphic routine
                     file from GRAPH.INC
                     to BUILDGR.INC
                     fixed multiple part making of
                     binary files
4/18/88   Ver 3.06   changes to drift correction logic
                     changed name of include files
                     independent-axis between profile
                     sensor movement
                     reset beam rest home position
                     after profiling
                     shortened can't locate profiler
                     error message
4/20/88   Ver 3.10   support 80x87 Numeric (Math)
                     Coprocessor
4/27/88   Ver 3.11   wait for final layer drawing to
                     be completed
                     * set up of laser-off control
                     through Menu program
                     * part start and completion times
                     logged to history file
                     ( * not for laser off )
5/04/88   Ver 3.12   text mode laser power-on
                     countdown display improved
                     fixed handling of keyboard during
                     DRAWING (waiting) period
                     (End) key for part name accepts
                     default name and places cursor at
                     end of line (for options input)
                     [works for graphics input only]
```

```
5/11/88   Ver 3.13   change GetTime to GetChipTime for
                     DOS
                     add /STAT Benchmark option
5/23/88   Ver 3.14   add thirty minute laser off entry
                     (again)
6/09/88              and put build.log in the local
                     directory
6/13/88   Ver 3.15   corrected STEREO and LASER error
                     descriptions
                     adapted for new LASER and STEREO
                     V3.15
6/20/88   Ver 3.16   support Part.PRM files
                     comment out auto laser wait
6/22/88   Ver 3.17   correct Delta processing of
                     part.PRM
6/24/88   Ver 3.18   don't use coprocessor. STEREO
                     conflicts
7/06/88   Ver 3.20   debugged Numeric Coprocessor
                     uses alternate LibTrunc and
                     LibRound routines
                     added /START and /STOP to option
                     line display
                     support latent /TEXT after
                     graphics started
                     ver # followed by C (Coprocessor)
                     or NC (No Coprocessor)
                     RealStr rounds instead of truncs
7/08/88   Ver 3.21   ESC at graphics Part Name prompt
                     aborts cleanly
                     Benchmark code made into a Unit
                     no errors reported for Abort and
                     Quit
                     better Option String order of
                     evaluation
                     relaxation period begins after
                     dip completes
                     pop-up menu created for showing
```

```
                        and altering options
                        improved graphics-based user I/O
                        routines
                        option line displayed in text
                        mode
                        DC +/- selected automatically
                        based on /NODRIFT option
     7/11/88  Ver 3.22  Ctrl-F6 resets NMI should the
                        80286 (AMD) hang up
                        took care of all the truncs and
                        rounds (we missed some)
     9/13/88  Ver 3.23  Add horseshoe path for drift
                        correction from sensor 1 to
                        sensor 2 movements
     9/14/88  Ver 3.24  correct options mixup and move
                        NoDrift flag logic (BLDMON)
-------------------------------------------------------*)

  {$R-,S-,I-,D-,T-,F-,V-,B-,L- }
  {$M 16384,0,100000 }

  { CoProcessor Support }
  {$IFOPT N+}
    {$DEFINE COP}
  {$ELSE}
    {$UNDEF COP}
  {$ENDIF}

uses Dos,       { standard Dos unit             }
     Crt,       { standard Crt unit             }
     LibNC,     { library of no-math coprocessor
                   routines }
     Bench,     { benchmark unit                }
     Graph,     { standard graph unit           }
     Drivers,   { graphics display drivers
                  extensions                     }
     Fonts;     { graphic font extensions       }
```

```
type
  {$IFDEF COP}
  FloatType = Single;
  {$ELSE}
  FloatType = Real;
  {$ENDIF}
  str4  = string[4];
  str6  = string[6];
  str10 = string[10];
  str16 = string[16];
  str40 = string[40];
  str80 = string[80];
  OpModeType = (TextOp,GraphicsOp);
  SpecialKeyType = (NoSpecial,Menu);
  FpPointerStruct = array[0..255] of Byte;
  FpPointerType   = ^FpPointerStruct;
  BeamProfileVarSensorType = record
    Xpos             : LongInt;   { inputs }
    Ypos             : LongInt;
    ScanStep         : Integer;
    BkgndThreshold   : Integer;
    InitialDelay     : Integer;
    RowToRowDelay    : Integer;
    ColToColDelay    : Integer;
    XYweight         : LongInt;   { outputs }
    Xweight          : LongInt;
    Yweight          : LongInt;
    TotalWeight      : LongInt;
    HiPoints         : Word;
    SaturationCount  : Integer;
    MaxSensorReading: Integer;
    BeamMap          : array[0..30,0..30] of Integer;
    RowWeight        : array[0..30] of Word;
    ColWeight        : array[0..30] of Word;
  end;
```

```
  BeamProfileVarType = array[1..3] of
BeamProfileVarSensorType;
  FindBeamVarSensorType = record
    Xval            : LongInt;    { inputs & updated }
    Yval            : LongInt;
    SpiralDelay     : Word;       { inputs }
    SpiralSideMax   : Integer;
    SpiralTries     : Integer;
    ScanStep        : Integer;
    BkgndThreshold  : Integer;
    NumCenterPasses : Integer;
    Xavg            : FloatType;       { outputs }
    Yavg            : FloatType;
  end;
  FindBeamVarType = array[1..3] of
FindBeamVarSensorType;
  FindMethodType = (NoMethod,NormalFind,Verify2MilHole);

  BenchActionsType =
(BENCH_BUILD_TOTAL,BENCH_BUILD_ANALYZE,BENCH
  _BUILD_LOADING,
    BENCH_BUILD_DRAWING,BENCH_BUILD_DIPPING,BENCH
_BUILD_LEVELING,
    BENCH_BUILD_PAUSED,BENCH_BUILD_SKIPPING,BENCH
_BUILD_SWEEPING);
  BuildActionsType =
(ACTION_LOADING,ACTION_DRAWING,ACTION_SURFACING,
    ACTION_DIPPING,ACTION_LEVELING,ACTION_PAUSED,
ACTION_SKIPPING,
    ACTION_ANALYZING,ACTION_PARTCOMPL,ACTION_SWEEPING);

const
  {$IFDEF COP}
  VersionId    = '3.24 C ';
  {$ELSE}
  VersionId    = '3.24 NC';
  {$ENDIF}
```

```
MSDOS          = $21;
LASERVECTOR    = $64;
STEREOVECTOR = $65;
BlockIdentifiers      = 'ZXYISHVABJCDEFGK';
ErrorFileName         = 'C:\SYSTEM.ERR';
SystemFileName        = 'C:\SLA1.SYS';
HomeDirectory         = 'C:\3DSYS\';
BlockIdFileName       = 'BLOCK.ID';
ParamFileName         = 'BUILD.PRM';
BeamParamFileName     = 'BEAM.PRM';
LastPartFileName      = 'PART.LST';
BuildHistoryFileName = 'BUILD.LOG';
CommandPath           = 'C:\COMMAND.COM';
CommandOpt            = '/C ';
BuildCommandsCount = 4;
BuildCommands: array[1..4] of str4 =
('ZW','ZD','ZV','ZA');
Ctrl C = ^C;
Bs     = ^H;
Tab    = ^I;
Lf     = ^J;
Cr     = ^M;
CtrlS = ^S;
Esc           = #27;
CrKey         = 13;
EscKey        = 27;
F1            = 1059;
F2            = 1060;
CtrlF6        = 1098;
CtrlF10       = 1103;
HomeKey       = 1071;
UpArrowKey    = 1072;
EndKey        = 1079;
DownArrowKey = 1080;
sp80 = '
                                    ';
DayOfWeek: String[21] = 'SunMonTueWedThuFriSat';
```

```
  Zel: array[1..12] of Integer =
(1,4,3,6,1,4,6,2,5,0,3,5);
  VectorRecordSize = 4096;
  PowerHistoryFileName   = 'C:\HISTORY\POWER.MON';
  ProfileHistoryFileName = 'C:\HISTORY\PROFILE.MON';
  Blank = '
                                  ';
  ColMax = 22;
  RowMax = 22;
  LaserAddr  = $300;
  NumSensors = 2;
  Timeout    = 60000;

  { Benchmark Constants }
  BenchmarkFileName = '\BENCH.ST';
  Build_Benchmark_Names : Array [0..8] of str16 =
                      ('Build Total   ',
                       'Build Analyze ',
                       'Build Loading ',
                       'Build Drawing ',
                       'Build Dipping ',
                       'Build Leveling',
                       'Build Paused  ',
                       'Build Skipping',
                       'Build Sweeping');

var
  FirstVar: Byte;  { must be first variable defined }
  Ch,key,key2: Char;
  SAVESENS,index,dummy,code,i,j,k,ordkey: Integer;
  WeekDay,Hour,Minu,Sec,Mon,Day,Year: Integer;

FirstLayer,SameBlock,MoveOnFlag,SkipFlag,PauseFlag,
DipFlag: Boolean;
  VectorFileIsBinary,SetForLaserOff: Boolean;
  Escape,STEREO_KeyCheck: Boolean;
```

```
  QSize,Layer,VPosition,NumLayers,LayersToBuild:
LongInt;
  ZPitch: FloatType;
  CurrentDate,CurrentTime: str16;
  ParamOptions,cmd,SaveCmd,LatentOverrideOptions:
String;

SingleBar,ControlLine,VectorLine,PartName,
DefaultPartName: str80;
  StartTimeStamp,EndTimeStamp: str40;
  BlockMnemonicsCount,x1,y1,x2,y2: Word;

VectorFileLoc,MaxLaserQueueBlockSize,LaserQueue
BlockSize: LongInt;
  Binary_X1,Binary_Y1,Binary_X2,Binary_Y2: Word;
  regs: Registers;
  LastPartFile,ControlFile,ErrorFile,HistoryFile: Text;
  VectorFile: File;  {untyped}
  XOffset,YOffset,ZAddress: LongInt;
  ParVectorType: array[0..100] of str10;
  ParStereoDef: array[0..100] of str80;
  BlockMnemonics: array[1..20] of str10;
  Offsets: array[0..50,1..2] of LongInt;
  LayerCount,NumXOffsets,NumYOffsets,NumOffsets: Word;
  ParCount,RelaxationTime: Word;
  LaserVersion,StereoVersion: str4;
  check1,check2: Boolean;

VisualPageNumber,VectorNumRead,BufIndex,ControlIndex,
BlockIndex: Integer;
  SaveSec,NewSec,DeltaSecs,OldDeltaSecs,ControlLayer,
VectorLayer: LongInt;
  GrDeltaX,GrDeltaY: LongInt;
  ControlType,VectorType: str10;
  DipDepth,ZTableVeloc,ZTableAccel: FloatType;
  SliceRes,ZSpacing,Thickness: FloatType;
  OnScreen,GraphDriver,GraphMode,Error: Integer;
```

```
  ViewPort: ViewPortType;
  OpMode: OpModeType;
  GrMinX,GrMinY,GrMaxX,GrMaxY: LongInt;
  OptionItem,RangeCount,BlockCount: Integer;
  BlockVectorCount,RangeZStart,RangeZEnd: LongInt;
  RangeZSpace,RangeCureDepth: Integer;
  ZStart,ZEnd: array[1..100] of LongInt;
    { all the ranges }
  ZSpace,CureDepth: array [1..100] of Integer;
    {                 }
  VectorTypes: array[1..100] of str6;
    { just for one range }
  BlockCmds: array[1..100] of str80;
    {                      }
  OldExitProc: Pointer;
  ExitSave: Pointer;

  { - OPTION VARIABLES - }
  ShowStuff,ZTableOff,LaserOff,NegZStage,SweepFlag:
Boolean;

TextFlag,StatFlag,NoDriftFlag,LayerCtrl,SwapXY,NegX,NegY
: Boolean;
  DemoOutOfVatDelay,StartZ,EndZ,PauseAfterLayer:
LongInt;
  XYScale,PartHeight: FloatType;

  { - PROFILING VARIABLES - }
  { beam arrays }
  Resistance,Temperature: array[0..100] of FloatType;
  { sensor arrays }
  SensorChannel,XSensorPos,YSensorPos,SensorBkgnd:
array[1..3] of Word;
  Power,Bk: array[1..3] of FloatType;
  Xsp,Ysp: array[1..3] of FloatType;
  SensorCalib: array[1..3,1..4] of FloatType;
  LastField: array[1..4] of Integer;
```

```
  { miscellaneous }
  Transparent,BeamFound: Boolean;
  XRest,YRest,TempChannel,TempCalPts: Word;
  GoodCount,PeakReading,Sensor,Select,SensorNum:
Integer;
  Xpos,Ypos,MonOldX,MonOldY: LongInt;
  Xaverage,Yaverage,TotWtAvg: FloatType;
  BeamProfileVariables: BeamProfileVarType;
  FindBeamVariables: FindBeamVarType;
  CalibDelays: array[1..3,1..3] of LongInt;
  SlewLimitMaxStep,SlewLimitDelay: Word;
  LastVar: Byte;  { must be last variable-to-be-zeroed
defined }

  VectorBuf: array[1..4100] of Char;

{-------------------------------------------------}

procedure OverText(Text: String;len: Integer); forward;
procedure PrintError(Text: String);            forward;
procedure LaserExecute;                        forward;

{ include various files }

   {$i stercmds.inc}   { STEREO binary command equates }
   {$i bldutil.inc}    { miscellaneous routines        }
   {$i bldkbd.inc}     { keyboard routines             }
   {$i bldgr.inc}      { screen graphics routines      }
   {$i bldopt.inc}     { pull-down options menu        }
   {$i bldtime.inc}    { date and time routines        }
   {$i bldzstg.inc}    { Z-stage elevator routines     }
   {$i bldlaser.inc}   { laser control routines        }
   {$i bldmon.inc}     { profile monitor routines      }
   {$i bldfile.inc}    { various file routines         }
{-------------------------------------------------}

{$F+} procedure BuildEnd; {$F-}
```

```
  begin
    if StatFlag then
BenchEnd(BenchmarkFileName,addr(Build_Benchmark_Names));

    ExitProc:=ExitSave;
  end;

procedure Message(Text: String);
begin
  if OpMode = TextOp then begin
    if whereX>1 then writeln;
    writeln(Text);
  end else begin
    MoveTo(3*8,24*14-2);
    OverText(Text,55);
  end;
end;

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
begin
  beep;
  Message('*** ' + Text);
end;

{-------------------------------------------------}
 procedure GraphVector;
var
  grx1,grx2,gry1,gry2,temp: Word;
begin
  if NegX then begin
    grx1 :=
LibRound((65535-x1-XOffset-GrMinX)/GrDeltaX*(77-45)*8);
    grx2 :=
LibRound((65535-x2-XOffset-GrMinX)/GrDeltaX*(77-45)*8);
```

```
  end else begin
    grx1 :=
LibRound((x1+XOffset-GrMinX)/GrDeltaX*(77-45)*8);
    grx2 :=
LibRound((x2+XOffset-GrMinX)/GrDeltaX*(77-45)*8);
  end;
  if NegY then begin
    gry1 :=
LibRound((65535-y1-YOffset-GrMinY)/GrDeltaY*(22-9)*14);
    gry2 :=
LibRound((65535-y2-YOffset-GrMinY)/GrDeltaY*(22-9)*14);
  end else begin
    gry1 :=
LibRound((y1+YOffset-GrMinY)/GrDeltaY*(22-9)*14);
    gry2 :=
LibRound((y2+YOffset-GrMinY)/GrDeltaY*(22-9)*14);
  end;
  if SwapXY then begin
    temp := grx1;
    grx1 := gry1;
    gry1 := temp;
    temp := grx2;
    grx2 := gry2;
    gry2 := temp;
  end;
  SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
  Line(grx1,gry1,grx2,gry2);
  SetViewPort(0,0,GetMaxX,GetMaxY,true);
end;

procedure SendVector(DrawMethod: Word);
type
  BinaryCommandType = array[1..11] of Byte;
const
  BinaryCommandMask: BinaryCommandType =
    (STEREO_JM,0,0,0,0,1,0,0,0,0,0);
var
```

```
  wd: Word;
  BinaryCommand: BinaryCommandType;
begin
  BinaryCommand := BinaryCommandMask;
  BinaryCommand[6] := DrawMethod;
  wd := LibTrunc(x1 + XOffset);
  BinaryCommand[2] := lo(wd);
  BinaryCommand[3] := hi(wd);
  wd := LibTrunc(y1 + YOffset);
  BinaryCommand[4] := lo(wd);
  BinaryCommand[5] := hi(wd);
  wd := LibTrunc(x2 + XOffset);
  BinaryCommand[7] := lo(wd);
  BinaryCommand[8] := hi(wd);
  wd := LibTrunc(y2 + YOffset);
  BinaryCommand[9]  := lo(wd);
  BinaryCommand[10] := hi(wd);
  SendBinaryLaserCommand(@BinaryCommand);
  if OpMode=GraphicsOp then GraphVector;
end;

procedure DrawVector;
begin
  SendVector(STEREO_MV);
end;

{-------------------------------------------------}

procedure DrawSectionRivet;
begin
  if true {always} then DrawVector
  else SendVector(STEREO_RV);
  if OpMode=GraphicsOp then GraphVector;
end;

{-------------------------------------------------}
```

```
procedure DrawSkinRivet;
begin
  if true (always) then DrawVector
  else SendVector(STEREO_RV);
  if OpMode=GraphicsOp then GraphVector;
end;

(=================================================

  BLOCK PROCESSING ROUTINES:                SLICE

  ZBlock:  Section Border                    V1
  XBlock:  Section X Hatching                V1
  YBlock:  Section Y Hatching                V1
  IBlock:  Section Angle Hatching            V18
8/26/87
  SBlock:  Up Facing Skin Border             V17
8/25/87
  HBlock:  Up Facing Skin X Fill             V17
  VBlock:  Up Facing Skin Y Fill             V17
  ABlock:  Up Facing Skin X Hatch            V17
  BBlock:  Up Facing Skin Y Hatch            V17
  JBlock:  Up Facing Skin Angle Hatch        V18
  CBlock:  Down Facing Skin Border           V17
  DBlock:  Down Facing Skin X Fill           V17
  EBlock:  Down Facing Skin Y Fill           V17
  FBlock:  Down Facing Skin X Hatch          V17
  GBlock:  Down Facing Skin Y Hatch          V17
  KBlock:  Down Facing Skin Angle Hatch      V18

  - New Mnemonic Set -

  BlockL    -  Layer id (no vectors)         V19
9/25/87
  BlockLB   -  Layer Border                  V19
  BlockLH   -  Layer Hatch                   V19
  BlockFUF  -  Flat Up skin Fill             V19
```

```
  BlockNFUB -  Near-Flat Up skin Border      V19a
9/28/87
  BlockNFUF -  Near-Flat Up skin Fill        V19
  BlockFDF  -  Flat Down skin Fill           V19
  BlockNFDB -  Near-Flat Down skin Border    V19
  BlockNFDH -  Near-Flat Down skin Hatch     V19
  BlockNFDF -  Near-Flat Down skin Fill      V19
  BlockFUB  -  Near-Flat Down skin Fill      V20
12/4/87
  BlockFDB  -  Near-Flat Down skin Fill      V20

-------------------------------------------------------}

procedure HandleBlockVector(VectorType: Integer);
begin
  case VectorType of
     1: DrawVector;         {ZBlock}
     2: DrawSectionRivet;   {XBlock}
     3: DrawSectionRivet;   {YBlock}
     4: DrawSectionRivet;   {IBlock}
     5: DrawVector;         {SBlock}
     6: DrawVector;         {HBlock}
     7: DrawVector;         {VBlock}
     8: DrawSkinRivet;      {ABlock}
     9: DrawSkinRivet;      {BBlock}
    10: DrawSkinRivet;      {JBlock}
    11: DrawVector;         {CBlock}
    12: DrawVector;         {DBlock}
    13: DrawVector;         {EBlock}
    14: DrawSkinRivet;      {FBlock}
    15: DrawSkinRivet;      {GBlock}
    16: DrawSkinRivet;      {KBlock}
    20: ;                   {BlockL}
    21: DrawVector;         {BlockLB}
    22: DrawSectionRivet;   {BlockLH}
    23: DrawSkinRivet;      {BlockFUF}
    24: DrawVector;         {BlockNFUB}
```

```
      25: DrawSkinRivet;      {BlockNFUF}
      26: DrawSkinRivet;      {BlockFDF}
      27: DrawVector;         {BlockNFDB}
      28: DrawSkinRivet;      {BlockNFDH}
      29: DrawSkinRivet;      {BlockNFDF}
      30: DrawVector;         {BlockFUB}
      31: DrawVector;         {BlockFDB}
    end;
  end;


{-------------------------------------------------}

{
  Break apart input line into two coordinates - (x1,y1)
and (x2,y2)
}
procedure GetCoords;
var x_1,y_1,x_2,y_2: FloatType;
begin
  if VectorFileIsBinary then begin
    if XYScale<>1.000 then begin
      x1 := LibTrunc(Binary_X1 * XYScale);
      y1 := LibTrunc(Binary_Y1 * XYScale);
      x2 := LibTrunc(Binary_X2 * XYScale);
      y2 := LibTrunc(Binary_Y2 * XYScale);
    end else begin
      x1 := LibTrunc(Binary_X1);
      y1 := LibTrunc(Binary_Y1);
      x2 := LibTrunc(Binary_X2);
      y2 := LibTrunc(Binary_Y2);
    end;
  end else begin  { ASCII file }
    BlockIndex := 1;
    UnitVal(x_1,VectorLine,BlockIndex);
    UnitVal(y_1,VectorLine,BlockIndex);
    UnitVal(x_2,VectorLine,BlockIndex);
```

```
    UnitVal(y_2,VectorLine,BlockIndex);
    if XYScale<>1.000 then begin
      x1 := LibTrunc(x_1 * XYScale);
      y1 := LibTrunc(y_1 * XYScale);
      x2 := LibTrunc(x_2 * XYScale);
      y2 := LibTrunc(y_2 * XYScale);
    end else begin
      x1 := LibTrunc(x_1);
      y1 := LibTrunc(y_1);
      x2 := LibTrunc(x_2);
      y2 := LibTrunc(y_2);
    end;
  end;
end;

{
  Process block entries until a new layer block header
appears
}
procedure ProcessBlockByLayer;
var
  n,ProcI,Ofs: Integer;
  Ctrl: str10;
  OrgSameBlock,BlockHadVectorsFlag: Boolean;
  OrgVectorLayer: Word;
  OrgVectorType: str10;
  OrgBlockVectorCount: LongInt;
begin
  { will return at end of block }
  n := 2;
  while (n<length(ControlType)) and not(ControlType[n]
in ['0'..'9'])
    do inc(n);
  ProcI := 0;
  if n=2 then
ProcI:=pos(ControlType[1],BlockIdentifiers);
  if ProcI=0 then begin
```

```
    Ctrl := copy(ControlType,1,n-1);
    for j:=1 to BlockMnemonicsCount do
      if Ctrl = BlockMnemonics[j] then ProcI := 19+j;
{ start at 20 }
  end;
  if ProcI>0 then begin
    if OpMode=TextOp then write(ControlType,' ')
    else begin
      MoveTo(28*8,13*14);
      OverText(ControlType,5);
    end;
    SetBlockDefaults;
    ProcessCommands(NoQuotes(copy(ControlLine,
  ControlIndex,199)));
    VPosition      := VectorFileLoc;
    OrgSameBlock   := SameBlock;
    OrgVectorLayer := VectorLayer;
    OrgVectorType  := VectorType;
    OrgBlockVectorCount := BlockVectorCount;
    for Ofs:=1 to NumOffsets do begin
      if Ofs>1 then begin
        VectorFileLoc := VPosition;
        ReSeekVectorFile;
      end;
      XOffset := Offsets[Ofs-1,1];
      YOffset := Offsets[Ofs-1,2];
      SameBlock := OrgSameBlock;
      VectorLayer := OrgVectorLayer;
      VectorType := OrgVectorType;
      BlockVectorCount := OrgBlockVectorCount;
      BlockHadVectorsFlag := false;
      while SameBlock do begin
        GetVectorEntry;
        if SameBlock then begin
          BlockHadVectorsFlag := true;
          GetCoords;
          HandleBlockVector(ProcI);
```

```
        end;
      end;
      if BlockHadVectorsFlag then LaserExecute;
    end;
  end else
    Message('Unrecognized block type '+VectorType+' in
layer '+WordStr(Layer));
end;

procedure ScanAndSendCommands(block: str6);
var
  i,len: Integer;
begin
  len := 0;
  i := 1;
  while (ParVectorType[i]<>block) and (i<=ParCount) do
inc(i);
  if i<=ParCount then begin
    ProcessCommands(ParStereoDef[i]);
    inc(len,length(ParStereoDef[i]));
  end;
  i := 1;
  while (VectorTypes[i]<>block) and (i<=BlockCount) do
inc(i);
  if i<=BlockCount then begin
    ProcessCommands(BlockCmds[i]);
    inc(len,length(BlockCmds[i]));
  end;
  if len > 0 then LaserExecute;
end;

{
  Process block entries until a new block entry appears
}
procedure ProcessBlockByRange;
var
  n,ProcI,Ofs: Integer;
```

```
  Ctrl: str10;
  OrgSameBlock,BlockHadVectorsFlag: Boolean;
  OrgVectorLayer: Word;
  OrgVectorType: str10;
  OrgBlockVectorCount: LongInt;
begin
  { will return at end of vector block }
  n := 2;
  while (n<length(VectorType)) and not(VectorType[n] in
['0'..'9'])
    do inc(n);
  ProcI := 0;
  if n=2 then
ProcI:=pos(VectorType[1],BlockIdentifiers);
  if ProcI=0 then begin
    Ctrl := copy(VectorType,1,n-1);
    for j:=1 to BlockMnemonicsCount do
      if Ctrl = BlockMnemonics[j] then ProcI := 19+j;
{ start at 20 }
  end;
  if ProcI>0 then begin
    if OpMode=TextOp then write(VectorType,' ')
    else begin
      MoveTo(28*8,13*14);
      OverText(VectorType,5);
    end;
    ScanAndSendCommands(VectorType);
    VPosition := VectorFileLoc;
    ControlType := VectorType;
    OrgSameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
    OrgVectorLayer := VectorLayer;
    OrgVectorType := VectorType;
    OrgBlockVectorCount := BlockVectorCount;
    for Ofs:=1 to NumOffsets do begin
      if Ofs>1 then begin
        VectorFileLoc := VPosition;
```

```
          ReSeekVectorFile;
        end;
        XOffset := Offsets[Ofs-1,1];
        YOffset := Offsets[Ofs-1,2];
        SameBlock := OrgSameBlock;
        VectorLayer := OrgVectorLayer;
        VectorType := OrgVectorType;
        BlockVectorCount := OrgBlockVectorCount;
        BlockHadVectorsFlag := false;
        while SameBlock and not EofVectorFile do begin
          GetVectorEntry;
          if SameBlock then begin
            BlockHadVectorsFlag := true;
            GetCoords;
            HandleBlockVector(ProcI);
          end;
        end;
        if BlockHadVectorsFlag then LaserExecute;
      end;
    end else
      Message('Unexpected block type '+VectorType+' in
layer '+WordStr(Layer));
end;

{-----------------------------------------------------

  Build Z-table dip routines

}
procedure DelayControl(DelayTime: Integer);
var
  ScreenH: Integer;
  line,temp: String;
  NewZ: Word;
begin
  ScreenH := whereX;
  DeltaSecs := 0;
```

```
  GetChipTime;
  SaveSec := Sec + 60*Minu + 3600*Hour;
  MoveOnFlag := false;
  SkipFlag   := false;
  if Layer >= PauseAfterLayer then begin
    PauseFlag := true;
    PauseAfterLayer := 65535;
  end else PauseFlag := false;
  while (DeltaSecs < DelayTime) and not MoveOnFlag do
begin
    WaitTest;
    if keypressed then begin
      GetBuildKey;
      if not HandleSpecialKeys then begin
        key := upcase(key);
        case key of
          'P': PauseFlag    := not PauseFlag;
          'C': begin
                 MoveOnFlag := true;    { force exit }
                 PauseFlag  := false;
               end;
          'S': begin
                 MoveOnFlag := true;
                 SkipFlag   := true;
                 PauseFlag  := false;
               end;
        end;
      end;
    end;
    if DeltaSecs <> OldDeltaSecs then begin
      OldDeltaSecs := DeltaSecs;
      temp := IntStr(DelayTime-DeltaSecs);
      if PauseFlag then temp := temp + '-P '
      else temp := temp + '   ';
      if OpMode=TextOp then begin
        gotoxy(ScreenH,whereY);
        write(temp);
```

```
        end else begin
          MoveTo(73*8,7*14);
          OverText(temp,5);
        end;
      end;
    end;  {while}
    gotoxy(ScreenH,whereY);
    if PauseFlag then temp := '0-P  ' else temp
:= '0';
    if OpMode=TextOp then begin
      gotoxy(ScreenH,whereY);
      write(temp);
    end else begin
      MoveTo(73*8,7*14);
      OverText(temp,5);
    end;
    if PauseFlag then begin   { handle pause }
      Action(ACTION_PAUSED);
      if OpMode=GraphicsOp then begin
        MoveTo(3*8,24*14-2);
        OutText('Building paused.  Press any key to
continue...      ');
      end;
      GetBuildKey;
      PauseFlag := false;
      if OpMode=GraphicsOp then begin
        MoveTo(3*8,24*14-2);
        OverText('',50);
      end;
    end;
    if (OpMode=TextOp) and (whereX>1) then writeln;
    if SkipFlag then begin   { handle layer skip }
      if OpMode=TextOp then begin
        write('Skip to what Z layer? ');
        readln(line);
      end else begin
        MoveTo(3*8,24*14-2);
```

```
        OutText('Skip to what Z layer? ');
        line := GetInput(8,Menu,'');
        MoveTo(3*7,24*14-2);
        OverText('',30);
        Action(ACTION_SKIPPING);
      end;
      if length(line)>0 then begin
        NewZ := LongVal(line,code);
        if NewZ>0 then StartZ := NewZ;
      end;
    end;
  end;

procedure WaitAndHandleKeys;
var
  clean: Boolean;
begin
  clean := false;
  STEREO_KeyCheck := false;   { don't let STEREO routine
handle keypresses }
  repeat
    SendLaserCommand('WI 1000');   { wait for background
laser control to idle }
    if keypressed then HandleKeyboardRequest
      else clean := true;
  until clean;
  STEREO_KeyCheck := true;   { allow STEREO routine to
handle keypressed }
end;

procedure DipAndWait;
var
  clean: Boolean;
begin
  if (OpMode = TextOp) and (whereX > 1) then writeln;
  Action(ACTION_DRAWING);
  WaitAndHandleKeys;
```

```
  ZMove('MN H+ V' +
RealStr(int(ZTableVeloc*10000.0)/10000,3)
    + ' A' + RealStr(int(ZTableAccel*10000.0)/10000,3));

  if DemoOutOfVatDelay<>0 then begin
    Action(ACTION_SURFACING);
    FormatZCommand(-PartHeight);
    DelayControl(DemoOutOfVatDelay);
    if SkipFlag then exit;
    FormatZCommand(PartHeight);
    delay(100);
    WaitForIdleZStage;   { wait for Z-stage to stop
moving }
  end;
  if SweepFlag then begin
    Action(ACTION_SWEEPING);
    Exec(CommandPath,CommandOpt + 'SWEEP');
  end else begin
    Action(ACTION_DIPPING);
    FormatZCommand(DipDepth);
  end;
  if DemoOutOfVatDelay<>0 then FormatZCommand(-DipDepth)

  else if not SweepFlag then begin
    if FirstLayer then begin
      FormatZCommand(Thickness-DipDepth);
      FirstLayer := false;
    end else if ZSpacing <> 0 then
      FormatZCommand((ControlLayer-Layer)/ZSpacing *
Thickness - DipDepth);
  end;
  delay(100);
  WaitForIdleZStage;
  Action(ACTION_LEVELING);
  if OpMode=GraphicsOp then begin
    MoveTo(67*8,7*14);
    OutText('TIME: '+WordStr(RelaxationTime));
```

```
    DelayControl(RelaxationTime);
    MoveTo(66*8,7*14);
    OverText('',10);
  end else DelayControl(RelaxationTime);
end;

{-------------------------------------------------------}
procedure HandleHeader(Line: String);
var
  Index: Integer;
begin
  Index := 2;
  while (Index<length(Line)) and (Line[Index]=' ')
    do inc(Index);
  UnitVal(SliceRes,Line,Index);
  UnitVal(ZSpacing,Line,Index);
  UnitVal(Thickness,Line,Index);
end;

procedure DetermineNumLayers;
var
  i: Integer;
  range: LongInt;
begin
  i := 1;
  range := ZStart[1];
  LayersToBuild := 0;
  while (i <= RangeCount) and (range <= EndZ) do begin
    if range >= StartZ then inc(LayersToBuild);
    if ZSpace[i]>0 then inc(range,ZSpace[i])
      else inc(range);
    if range > ZEnd[i] then begin
      inc(i);
      range := ZStart[i];
    end;
  end;
end;
```

```
procedure DriveByLayers;
begin
{
   determines the next block to execute
   can skip blocks on specific layers
   handles dipping, screen updates
}
  while (ControlLine <> '') and (ControlLayer <= EndZ)
do begin
    if ControlLayer < StartZ then begin
      BlockSkip;
      GetLayerControlLine;
     end else if ControlType='#TOP' then begin
      if DipFlag then begin
        DipAndWait;
        DipFlag := false;
      end;
      MonitorLaserBeamDrift;
      if not SkipFlag then begin
        Layer := ControlLayer;
        if OpMode=TextOp then write(Layer,': ')
        else begin
          MoveTo(14*8,13*14);
          OverText(WordStr(Layer),5);

SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
          ClearViewPort;
          SetViewPort(0,0,GetMaxX,GetMaxY,true);
          MoveTo(7*8,7*14);
          inc(LayerCount);
          if LayersToBuild < 65535 then
            OverText(WordStr(LayerCount)+' OF
'+WordStr(LayersToBuild),13)
              else OverText(WordStr(LayerCount),13);
        end;
        Action(ACTION_DRAWING);
```

```
        SetBlockDefaults;
        cmd :=
NoQuotes(copy(ControlLine,ControlIndex,199));
        if length(cmd)>0 then begin
          ProcessCommands(cmd);
          LaserExecute;
        end;
      end;
      SkipFlag := false;
      GetLayerControlLine;
    end else if ControlType='#BTM' then begin
      SetBlockDefaults;
      cmd :=
NoQuotes(copy(ControlLine,ControlIndex,199));
      if length(cmd)>0 then begin
        ProcessCommands(cmd);
        LaserExecute;
      end;
      DipFlag := true;
      GetLayerControlLine;
{ will skip blocks for those *.L block entries edited
out }
    end else begin
      if SameBlock then begin
        ProcessBlockByLayer;
        GetLayerControlLine;
      end else GetVectorEntry;
    end;
  end;
end;
procedure DriveByRanges;
begin
{
   utilizes range information
   can't skip blocks on specific layers
   doesn't support critical areas
   handles dipping, screen updates
```

```
}
  while (not EofVectorFile) and (VectorLayer <= EndZ) do
begin
    while VectorLayer > RangeZEnd do ReadRange;
    if VectorLayer < StartZ then begin
      BlockSkip;
      DetermineNumLayers;   { adjust layer m of n values
}
      inc(LayersToBuild,LayerCount);
    end else begin
      ControlLayer := VectorLayer;  { dip based on
ControlLayer - Layer }
      if DipFlag then begin
        DipAndWait;
        DipFlag := false;
      end;
      MonitorLaserBeamDrift;
      if VectorLayer >= StartZ then begin
        Layer := ControlLayer;
        if OpMode=TextOp then write(Layer,': ')
        else begin
          MoveTo(14*8,13*14);
          OverText(WordStr(Layer),5);

SetViewPort(45*8+4,9*14+4,77*8+4,22*14+4,true);
          ClearViewPort;
          SetViewPort(0,0,GetMaxX,GetMaxY,true);
          MoveTo(7*8,7*14);
          inc(LayerCount);
          if LayersToBuild < 65535 then
            OverText(WordStr(LayerCount)+' OF
'+WordStr(LayersToBuild),13)
              else OverText(WordStr(LayerCount),13);
        end;
        Action(ACTION_LOADING);
        ScanAndSendCommands('#TOP');
```

```
        while (ControlLayer = VectorLayer) and not
EofVectorFile
          do ProcessBlockByRange;
        ScanAndSendCommands('#BTM');
        DipFlag := true;
      end;
    end;
  end;
end;

procedure PartMake;
var
  cmd,Text: String;
begin
  assign(LastPartFile,LastPartFileName);
  {$I-}
  rewrite(LastPartFile);
  {$I+}
  if IOresult = 0 then begin
    writeln(LastPartFile,PartName);
    close(LastPartFile);
  end;

(**** COMMENT OUT
  if not LaserOff then begin
    if not HardwareLaserOn then begin
      Exec(CommandPath,CommandOpt + 'SERVICE LASER ON
NOW/i/t');
      Message('Waiting for laser to power up.');
      minu := 15;
      sec  := 0;
      key  := ' ';
      while (minu >= 0) and (key <> 'C') do begin
        while (sec >= 0) and (key <> 'C') do begin
          Text := PadSp(minu) + ':' + LZero(sec);
          if OpMode = TextOp then write(Cr,Text)
          else begin
```

```
              MoveTo(60*8,24*14-2);
              OverText(Text,5);
            end;
            delay(1000);
            dec(sec);
            if keypressed then HandleKeyboardRequest;
            key := upcase(key);
          end;
          dec(minu);
          sec := 59;
        end;
        if OpMode = GraphicsOp then begin
          MoveTo(3*8,24*14-2);
          OverText('',70);
        end else writeln;
      end;
    end;
**** END COMMENT OUT}

    if (NumOffsets > 1) and (OpMode = TextOp) then begin
      writeln(IntStr(NumOffsets),' multiples being made at
once.');
      writeln;
    end;
    SendLaserCommand('HR;MD BL +');
    if NoDriftFlag then SendLaserCommand('DC -')
      else SendLaserCommand('DC +');
    FirstLayer := true;
    repeat
      LayerCount := 0;
      SendLaserCommand('CL;EC');
      LaserQueueBlockSize := 0;

  {
     read table headers
  }
      if LayerCtrl then begin
```

```
      repeat
        GetLayerControlLine;
        if ControlLine[1]='*' then
HandleHeader(ControlLine);
      until ControlLine[1]<>'*';
      LayersToBuild := NumLayers;
    end else begin
      RangeCount := 0;
      ControlLine := 'BEGIN';
      while ControlLine<>'' do begin   { read all ranges
in }
        GetRangeControlLine;
        if ControlLine[1]='*' then
HandleHeader(ControlLine);
        if ControlLine[1] in ['0'..'9'] then begin
          inc(RangeCount);
          Index := 1;
          UnitValInt     (ZStart
[RangeCount],ControlLine,Index);
          UnitValInt     (ZEnd
[RangeCount],ControlLine,Index);
          UnitValShortInt(ZSpace
[RangeCount],ControlLine,Index);

UnitValShortInt(CureDepth[RangeCount],ControlLine,Index)
;
        end;
      end;
      close(ControlFile);
      DetermineNumLayers;   { determine # of layers from
range data }
      reset(ControlFile);
      repeat
        GetRangeControlLine;
      until (ControlLine='') or (ControlLine[1]<>'*');
      ReadRange;   { load in block data for first range
}
```

```
      Layer := ZStart[1];
    end;

    BufIndex := 1;
    if VectorFileIsBinary then VectorFileLoc := 1
      else VectorFileLoc := 0;
    BlockVectorCount := 0;

BlockRead(VectorFile,VectorBuf,VectorRecordSize,VectorNu
mRead);
    repeat
      GetVectorEntry;
      if VectorLine[1]='*' then
HandleHeader(VectorLine);
    until VectorLine[1]<>'*';
    Layer := -1;

{
  update Action Screen
}
    if OpMode=GraphicsOp then begin
      MoveTo(1*8,7*14);
      if LayersToBuild<65535 then OutText('LAYER 1 OF
'+WordStr(LayersToBuild))
        else OutText('LAYER 1');
      MoveTo(14*8,12*14);
      OutText(WordStr(StartZ));
      MoveTo(14*8,14*14);
      OutText(WordStr(EndZ));
    end;

    if LayerCtrl then DriveByLayers   { decide on
control routine }
      else DriveByRanges;

    close(ControlFile);
    close(VectorFile);
```

```
    if DemoOutOfVatDelay<>0 then OpenFiles;
  until DemoOutOfVatDelay = 0;
  WaitAndHandleKeys;
end;

{-------------------------------------------------}
procedure CheckForOptions(var cmd: string);
var
  i,oi,j: Integer;
begin
  i := 1;
  while (i <= length(cmd)) and (cmd[i] = ' ') do inc(i);

  cmd := upper(copy(cmd,i,255));
  i := pos('/TEXT',cmd);   { check for /TEXT text
operation }
  if i>0 then begin
    OpMode := TextOp;
    TextFlag := true;
    delete(cmd,i,5);
  end;
  i := pos('/START',cmd);   { check for /START Z-layer
start }
  if i>0 then begin
    oi := i;
    inc(i,6);
    while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
    j := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),StartZ,code);
    delete(cmd,oi,j-oi);
  end;
  i := pos('/STOP',cmd);   { check for /END Z-layer end
}
  if i>0 then begin
    oi := i;
    inc(i,5);
    while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
```

```
    j := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),EndZ,code);
    delete(cmd,oi,j-oi);
  end;
  i := pos('/PAUSE',cmd);   { check for /PAUSE option }
  if i>0 then begin
    oi := i;
    inc(i,6);
    while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
    j := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),PauseAfterLayer,code);
    delete(cmd,oi,j-oi);
  end;
  i := pos('/SWAPXY',cmd);  { swap X & Y coordinates }
  if i>0 then begin
    SwapXY := true;
    delete(cmd,i,7);
  end;
  i := pos('/X',cmd);  { X-axis orientation }
  if i>0 then begin
    NegX := (cmd[i+2]='-');
    delete(cmd,i,3);
  end;
  i := pos('/Y',cmd);  { Y-axis orientation }
  if i>0 then begin
    NegY := (cmd[i+2]='-');
    delete(cmd,i,3);
  end;
  i := pos('/SWEEP',cmd);     { check for Sweeper Arm
option }
  if i>0 then beg
    SweepFlag := ue;
    ZTableOff := true;
    delete(cmd,i,6);
  end;
  i := pos('/STAT',cmd);      { check for Benchmark
Stats option }
```

```
  if i>0 then begin
    StatFlag := True;
    delete(cmd,i,5);
  end;
  i := pos('/SHOW',cmd);
  if i>0 then begin   { check for /SHOW Show Stuff flag
}
    ShowStuff := true;
    delete(cmd,i,5);
  end;
  i := pos('/ZOFF',cmd);   { check for /ZOFF No
Z-Control flag }
  if i>0 then begin
    ZTableOff := true;
    delete(cmd,i,5);
  end;
  i := pos('/LOFF',cmd);   { check for /LOFF No Laser
flag }
  if i>0 then begin
    LaserOff := true;
    delete(cmd,i,5);
  end;
  i := pos('/NEGZ',cmd);   { check for /NEGZ Neg-Z Stage
flag }
  if i>0 then begin
    NegZStage := true;
    delete(cmd,i,5);
  end;
  i := pos('/HEIGHT',cmd);   { check for /HEIGHT flag }
  if i>0 then begin
    oi := i;
    inc(i,7);
    while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
    j := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),PartHeight,code);
    delete(cmd,oi,j-oi);
  end;
```

```
  i := pos('/DEMO',cmd);    { check for /DEMO flag }
  if i>0 then begin
    oi := i;
    inc(i,5);
    while (i<length(cmd)) and (cmd[i]=' ') do inc(i);
    j := FindStr(i,cmd,' /');
    val(copy(cmd,i,j-i),DemoOutOfVatDelay,code);
    delete(cmd,oi,j-oi);
  end;
  i := pos('/LC',cmd);    { check for Layer Control
option }
  if i>0 then begin
    LayerCtrl := true;
    delete(cmd,i,3);
  end;
  i := pos('/NODRIFT',cmd);    { check for no drift
correction option }
  if i>0 then begin
    NoDriftFlag := true;
    delete(cmd,i,8);
  end;
end;

procedure ResetOptions;
var
  i,oi,j: Integer;
begin
    OpMode    := GraphicsOp;
    TextFlag  := false;
    SwapXY    := false;
    NegX      := false;
    NegY      := false;
    SweepFlag := false;
    ZTableOff := false;
    StatFlag  := false;
    ShowStuff := false;
    LaserOff  := false;
```

```
    NegZStage := false;
    DemoOutOfVatDelay := 0;
    LayerCtrl := false;
    NoDriftFlag := false;
    CheckForOptions(Savecmd);
      { originally input options }
end;

procedure ReadBlockIdFile;
var
  BlockIdFile: text;
  BlockIdLine: string;
  Group: Integer;
  Block: String;
begin
  BlockMnemonicsCount := 0;
  {$I-}
  assign(BlockIdFile,BlockIdFileName);
  reset(BlockIdFile);
  if IOresult<>0 then begin
    assign(BlockIdFile,HomeDirectory+BlockIdFileName);
    reset(BlockIdFile);
    if IOresult<>0 then begin
      writeln('Can''t find Block Id file.');
      halt(2);
    end;
  end;
  {$I+}
  while not eof(BlockIdFile) do begin
    readln(BlockIdFile,BlockIdLine);
    if (length(BlockIdLine)>0) and (BlockIdLine[1]<>'!')
then begin
      index := 1;
      UnitValShortInt(Group,BlockIdLine,index);
      UnitStr(Block,BlockIdLine,index);
      if Group=0 then begin
        inc(BlockMnemonicsCount);
```

```
        BlockMnemonics[BlockMnemonicsCount] :=
upper(Block);
      end;
    end;
  end;
  close(BlockIdFile);
end;

procedure GetXOffsets(line: string);
begin
  NumXOffsets := 0;
  repeat
    Offsets[NumXOffsets,1] := LongVal(line,code);
    inc(NumXOffsets);
    j := FindStr(1,line,', ');
    if (j=length(line)+1) or (line[j]=' ') then exit;
    line := copy(line,j+1,199);
  until NumXOffsets>50;
end;

procedure GetYOffsets(line: string);
begin
  NumYOffsets := 0;
  repeat
    Offsets[NumYOffsets,2] := LongVal(line,code);
    inc(NumYOffsets);
    j := FindStr(1,line,', ');
    if (j=length(line)+1) or (line[j]=' ') then exit;
    line := copy(line,j+1,199);
  until NumYOffsets>50;
end;

procedure GetXY(line: String;var Xvar,Yvar: LongInt);
begin
  Xvar := LongVal(line,code);
  Yvar := LongVal(copy(line,code+1,255),code);
end;
```

```
procedure GetBuildParams(var ParamFile : Text);
var
  count: Integer;
  cmd,ParamLine: String;
begin
  count := 0;
  while (not eof(ParamFile)) and (count<=8) do begin
    readln(ParamFile,ParamLine);
    if (length(ParamLine)>0) and (ParamLine[1]<>'!')
then begin
      case count of
        0: begin
             ParamOptions := upper(NoQuotes(ParamLine));

             if length(ParamOptions)=0 then ParamOptions
:= 'NONE';
             CheckForOptions(ParamLine);
           end;
        1: ZAddress := IntVal(ParamLine,code);
        2: ZPitch  := RealVal(ParamLine,code);
        3: XYScale := RealVal(ParamLine,code);
        4: GetXOffsets(ParamLine);
        5: GetYOffsets(ParamLine);
        6: MaxLaserQueueBlockSize :=
IntVal(ParamLine,code);
        7: GetXY(ParamLine,GrMinX,GrMinY);
        8: GetXY(ParamLine,GrMaxX,GrMaxY);
      end;
      inc(count);
    end;
  end;
  NumOffsets := NumXOffsets;
  if NumOffsets > NumYOffsets then NumOffsets :=
NumYOffsets;
  ParCount := 0;
  while not eof(ParamFile) do begin
```

```
    cmd := '';
    repeat
      repeat
        if eof(ParamFile) then begin
          SystemError('Error - Unexpected end of
BUILD.PRM file!');
          halt(4);
        end;
        readln(ParamFile,ParamLine);
        i := 1;   { clean up input, zap some white space
}
        while (i<length(ParamLine)) and (ParamLine[i] in
[Tab,' ']) do inc(i);
        delete(ParamLine,1,i-1);
      until (length(ParamLine)>0) and
(ParamLine[1]<>'!');
      j := pos('!',ParamLine);
      if j>0 then begin
        dec(j);
        while (j>0) and (ParamLine[j] in [Tab,' ']) do
dec(j);
        delete(ParamLine,j+1,199);
      end;
      cmd := cmd + ParamLine;
    until cmd[length(cmd)]<>';';
    inc(ParCount);
    i := FindStr(1,cmd,',');
    ParVectorType[ParCount] := copy(cmd,1,i-1);
    ParStereoDef[ParCount]  :=
NoQuotes(copy(cmd,i+1,199));
  end;
  close(ParamFile);
end;

procedure ComputeDelta;
begin
  GrDeltaX := GrMaxX-GrMinX;
```

```
  if GrDeltaX < 10 then GrDeltaX := 10;
  GrDeltaY := GrMaxY-GrMinY;
  if GrDeltaY < 10 then GrDeltaY := 10;
end;

procedure ReadBuildParamsFile;
var
  ParamFile: text;
  count: Integer;
  cmd,ParamLine: String;
begin
  Count := 0;
  repeat
    Inc(Count);
    Case Count of
      1 : Assign(ParamFile,ParamFileName);
      2 : Assign(ParamFile,HomeDirectory+ParamFileName);

    end;
    {$I-} reset(ParamFile); {$I+}
  until (Count=2) or (IOResult=0);
  if IOresult<>0 then begin
    writeln('Can''t find Build Parameter file.');
    halt(3);
  end;
  GetBuildParams(ParamFile);
  ComputeDelta;
end;

procedure ReadPartParamFile;
var
  ParamFIle : Text;
begin
  Assign(ParamFile,PartName+'.PRM');
  {$I-} reset(ParamFile); {$I+}
  if IOresult=0 then begin
    ResetOptions; { re-reads saveCmd at the end }
```

```
    GetBuildParams(ParamFile);
    ComputeDelta;
  end else ReadBuildParamsFile;
end;

procedure ReadLastPartFile;
var
   i : integer;
begin
  DefaultPartName := '';
  assign(LastPartFile,LastPartFileName);
  {$I-}
  reset(LastPartFile);
  {$I+}
  if IOresult <> 0 then exit;
  readln(LastPartFile,DefaultPartName);
  DefaultPartName := upper(copy(DefaultPartName,1,50));
  close(LastPartFile);
{}  i := pos('.',DefaultPartName);
  if i=0 then i := length(DefaultPartName)+1;
  PartName := upper(copy(DefaultPartName,1,i-1));
  ReadPartParamFile;
  PartName := '';
end;

procedure ShowFiles(FileSpec: string);
var
  FileCount: Integer;
  DirInfo: SearchRec;
begin
  FileCount := 0;
  FindFirst(FileSpec,AnyFile,DirInfo);
  if DosError<>0 then writeln('- none -')
  else begin
    while DosError=0 do begin
      if FileCount>0 then write(spc(11-((whereX-1) mod
10+1)));
```

```
      inc(FileCount);

write(copy(DirInfo.Name,1,pos('.',DirInfo.Name)-1));
      FindNext(DirInfo);
    end;
    writeln;
  end;
end;

procedure GetPartFileNameText;
var
  remX: Integer;
begin
  if PartName = '' then begin
    writeln('Directory Listing of Part Files:');
    writeln;
    ShowFiles('*.V');
    writeln;
    write('Part Name: ');
    remX := whereX;
    write(DefaultPartName);
    gotoxy(remX,whereY);
    readln(PartName);
    if PartName = '' then PartName := DefaultPartName;
  end;
  PartName := upper(PartName);
  i := pos('/',PartName);
  if i>0 then begin
    LatentOverrideOptions := copy(PartName,i,255);
    PartName[0] := chr(i-1);
  end else LatentOverrideOptions := '';
end;

procedure GetPartFileNameGr;
var
  name: str16;
  DirInfo: SearchRec;
```

```
  FileCount,remX: Integer;
begin
  if length(PartName)=0 then begin
    FileCount := 0;
    MoveTo(1*8,11*14);
    OutText('Parts Directory Listing:');
    FindFirst('*.V',AnyFile,DirInfo);
    MoveTo(1*8,12*14);
    if DosError<>0 then OutText('- NONE -')
    else begin
      while (DosError = 0) and (GetY < 23*14) do begin
        inc(FileCount);
        name :=
copy(DirInfo.Name,1,pos('.',DirInfo.Name)-1);
        name := name + spc(12-length(name));
        OutText(name);
        if GetX>8*60 then MoveTo(1*8,GetY+14);
        FindNext(DirInfo);
      end;
      writeln;
    end;
    MoveTo(1*8,22*14);
    SetColor(Green);
    OutText('Part Name: ');
    SetColor(Cyan);
    PartName :=
upper(GetInput(50,Menu,DefaultPartName));
    if Escape then Quit;
    i := pos('/',PartName);
    if i>0 then begin
      LatentOverrideOptions := copy(PartName,i,255);
      PartName[0] := chr(i-1);
    end else LatentOverrideOptions := '';
  end;
end;

procedure MakeTitle;
```

```
begin
  ClearViewPort;
  SetColor(Cyan);
  DrawBorder;
  MoveTo(10,5);
  SetTextStyle(TriplexFont,HorizDir,4);
  OutText('3D Systems');
  MoveTo(13,40);
  SetTextStyle(SmallFont,HorizDir,6);
  OutText('Sylmar, California');
  MoveTo(295,9);
  SetTextStyle((SansSerifFont)TriplexFont,HorizDir,4);
  OutText('BUILD');
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo(265,45);
  OutText('Copyright (C) 1988');
  MoveTo(64*8,1*14+7);
  OutText('SLA-1');
  MoveTo(64*8,2*14+7);
  OutText('Version '+VersionId);
  SetLineStyle(SolidLn,0,ThickWidth);
  Line(0,65,639,65);
  SetLineStyle(SolidLn,0,NormWidth);
end;

function OptionsInString: String;
var
  OptionStr: String;
begin
  OptionStr := '';
  if NegX          then OptionStr := OptionStr + '/X-';
  if NegY          then OptionStr := OptionStr + '/Y-';
  if SwapXY        then OptionStr := OptionStr +
'/SWAPXY';
  if ZTableOff     then OptionStr := OptionStr + '/ZOFF';
```

```
  if LaserOff     then OptionStr := OptionStr + '/LOFF';

  if NegZStage    then OptionStr := OptionStr + '/NEGZ';

  if NoDriftFlag  then OptionStr := OptionStr +
'/NODRIFT';
  if StartZ > 0   then OptionStr := OptionStr + '/START
' + WordStr(StartZ);
  if EndZ < 65535 then OptionStr := OptionStr + '/STOP '
+ WordStr(EndZ);
  if PauseAfterLayer<65535 then OptionStr := OptionStr +
'/PAUSE'
                                                    +
LongStr(PauseAfterLayer);
  if LayerCtrl    then OptionStr := OptionStr + '/LC';
  if SweepFlag    then OptionStr := OptionStr +
'/SWEEP';
  if StatFlag     then OptionStr := OptionStr + '/STAT';

  if DemoOutOfVatDelay<>0 then OptionStr := OptionStr +
'/DEMO';
  if ShowStuff    then OptionStr := OptionStr + '/S';
  OptionsInString := OptionStr;
end;

procedure MakeParamScreen;
var
  i: Integer;
begin
  MakeTitle;
  Line(0,6*14+4,639,6*14+4);
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo(1*8,7*14);
  OutText('Options: '+OptionsInString);
  Line(0,8*14+4,639,8*14+4);
  MoveTo(1*8,9*14);
  OutText('XY-Scale: '+RealStr(XYScale,4));
```

```
  MoveTo(55*8,9*14);
  OutText('# Parts:       X     Y');
  Line(0,10*14+4,639,10*14+4);
  Line(68*8-4,8*14+4,68*8-4,23*14+4);
  Line(74*8-4,8*14+4,74*8-4,23*14+4);
  Line(54*8-4,8*14+4,54*8-4,10*14+4);
  Line(0,21*14+4,68*8-4,21*14+4);
  Line(0,23*14+4,639,23*14+4);
  MoveTo(64*8,9*14);
  OutText(WordStr(NumOffsets));
  i := 0;
  while (i<12) and (i<NumOffsets) do begin
    MoveTo(68*8,(11+i)*14);
    OutText(LongStr(Offsets[i,1]));
    MoveTo(74*8,(11+i)*14);
    OutText(LongStr(Offsets[i,2]));
    inc(i);
  end;
end;

procedure MakeActionScreen;
var
  H,V: Char;
  NegHorz,NegVert: Boolean;
begin
  MakeTitle;
  Line(0,6*14+4,639,6*14+4);
  SetTextStyle(DefaultFont,HorizDir,1);
  MoveTo(32*8,7*14);
  OutText('ACTION: ');
  SetColor(Green);
  OutText('INITIALIZING');
  SetColor(Cyan);
  Line(0,8*14+4,639,8*14+4);
  MoveTo(1*8,5*14+3);
  OutText('PART '+PartName);
  MoveTo(64*8,5*14+3);
```

```
OutText('CONTROL: ');
if LayerCtrl then OutText('LAYER')
  else OutText('RANGE');
MoveTo(1*8,9*14);
OutText('START TIME: '+StartTimeStamp);
MoveTo(3*8,10*14);
OutText('END TIME: '+EndTimeStamp);
Line(0,11*14+4,35*8+4,11*14+4);
MoveTo(1*8,12*14);
OutText('START LAYER:');
MoveTo(4*8,13*14);
OutText('ON LAYER:');
MoveTo(3*8,14*14);
OutText('END LAYER:');
Line(0,15*14+4,35*8+4,15*14+4);
MoveTo(28*8,12*14);
OutText('BLOCK');
Line(35*8+4,8*14+4,35*8+4,15*14+4);
Line(24*8+4,11*14+4,24*8+4,15*14+4);
Rectangle(45*8+4-1,9*14+4-1,77*8+4+1,22*14+4+1);
Line(0,23*14+4,639,23*14+4);
{ draw axes orientation arrows }
NegVert := NegY;
NegHorz := NegX;
V := 'Y';
H := 'X';
if SwapXY then begin
  NegVert := NegX;
  NegHorz := NegY;
  V := 'X';
  H := 'Y';
end;
case NegVert of
  false: case NegHorz of
           false: begin

Line(38*8+4,10*14+4,38*8+4,12*14+4);
```

```
Line(38*8+4,10*14+4,42*8+4,10*14+4);
                        MoveTo(43*8,10*14);
                        OutText(H);
                        MoveTo(38*8,13*14-4);
                        OutText(V);
                      end;
              true : begin

Line(42*8+4,10*14+4,42*8+4,12*14+4);

Line(38*8+4,10*14+4,42*8+4,10*14+4);
                        MoveTo(42*8,13*14-4);
                        OutText(V);
                        MoveTo(37*8,10*14);
                        OutText(H);
                      end;
           end;
    true : case NegHorz of
             false: begin

Line(38*8+4,10*14+4,38*8+4,12*14+4);

Line(38*8+4,12*14+4,42*8+4,12*14+4);
                        MoveTo(38*8,9*14+4);
                        OutText(V);
                        MoveTo(43*8,12*14);
                        OutText(H);
                      end;
              true : begin

Line(42*8+4,10*14+4,42*8+4,12*14+4);

Line(38*8+4,12*14+4,42*8+4,12*14+4);
                        MoveTo(37*8,12*14);
                        OutText(H);
                        MoveTo(42*8,9*14+4);
```

```
                        OutText(V);
                      end;
          end;
  end;
end;

procedure BuildWithText;
begin
  GetPartFileNameText;
  ReadPartParamFile;
  CheckForOptions(LatentOverrideOptions);
  if OpMode <> TextOp then begin
     exit;
  end;

  if (NumOffsets < 1) or (PartName = '') then halt(0);
  if StatFlag then begin
    BenchInit(BenchmarkFileName);
    BenchStart(ord(BENCH_BUILD_TOTAL));
  end;
  writeln;
  OpenFiles;
  writeln('Starting part-making process...');
  writeln;
  writeln('Options: ',OptionsInString);
  writeln;
  GetTimeStamp;
  StartTimeStamp := CurrentDate + ' ' + CurrentTime;
  Log('START ' + StartTimeStamp + ' ' + PartName);
  InitZTable;
  PartMake;
  writeln;
  SystemError('Part completed.');
  writeln;
  writeln('Start Time: ',StartTimeStamp);
  GetTimeStamp;
  EndTimeStamp := CurrentDate + ' ' + CurrentTime;
```

```
  Log('COMPL ' + EndTimeStamp + ' ' + PartName);
  writeln('  End Time: ',EndTimeStamp);
  if (not LaserOff) then begin
     Exec(CommandPath,CommandOpt + 'SERVICE LASER OFF
+30/i/t');
     SetForLaserOff := true;
  end;
end;

procedure BuildWithGraphics;
begin
  InstallGraphics;
  InitOptionsMenu;
  BuildImage;
  MakeParamScreen;
  ShowImage;
  GetPartFileNameGr;
  ReadPartParamFile;
  CheckForOptions(LatentOverrideOptions);
  if OpMode = TextOp then begin
    CloseGraph;
    RestoreCrtMode;
    BuildWithText;
  end else begin
    MakeParamScreen;
    if (NumOffsets < 1) or (PartName = '') then begin
      CloseGraph;
      TextMode(C80);
      halt(0);
    end;
    if StatFlag then begin
      BenchInit(BenchmarkFileName);
      BenchStart(ord(BENCH_BUILD_TOTAL));
    end;
    OpenFiles;
    BuildImage;
    GetTimeStamp;
```

```
    StartTimeStamp := CurrentDate + ' ' + CurrentTime;
    Log('START ' + StartTimeStamp + ' ' + PartName);
    MakeActionScreen;
    ShowImage;
    InitZTable;
    PartMake;
    GetTimeStamp;
    EndTimeStamp := CurrentDate + ' ' + CurrentTime;
    Log('COMPL ' + EndTimeStamp + ' ' + PartName);
    MoveTo(13*8,10*14);
    OutText(EndTimeStamp);
    Action(ACTION_PARTCOMPL);
    if (not LaserOff) then begin
      Exec(CommandPath,CommandOpt + 'SERVICE LASER OFF
+30/i/t');
      SetForLaserOff := true;
    end;
    CloseGraph;
  end;
end;


{  -  -  -  M A I N  -  -  -  }

begin
  CheckBreak := false;
  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);    {
clear variables }
  ExitSave:=ExitProc;
  ExitProc:=@BuildEnd;
  SingleBar := many(chr(196),45);
  writeln;
  writeln(SingleBar);
  writeln('  BUILD, Stereolithography Part-Builder');
  writeln('  Version ',VersionId);
  writeln('  Copyright (C) 1988 by 3D Systems, Inc.');
```

```
  writeln('  3D Systems Laser Stereolithography
System');
  writeln(SingleBar);
  writeln;
  OpMode := TextOp;
  check1 := LaserPresent;
  check2 := StereoPresent;
  if not (check1 and check2) then halt(1);
  OpMode := GraphicsOp;
  StartZ          := 0;
  EndZ            := 65535;
  NumLayers       := 65535;
  PauseAfterLayer := 65535;
  STEREO_KeyCheck := true;
  SAVESENS := 1;

  ReadBlockIdFile;
  ReadBuildParamsFile;
  ReadBeamParamsFile;
  ReadLastPartFile;
  SetupMonitoring;
  ReadCommandLine(cmd);    { process overriding command
line }
  i := pos('/',cmd);
  if i=0 then i := length(cmd)+1;
  PartName := upper(copy(cmd,1,i-1));
  SaveCmd := upper(copy(cmd,i,255));
  cmd := SaveCmd;
  CheckForOptions(cmd);
  if OpMode= TextOp then BuildWithText;
  if OpMode<>TextOp then BuildWithGraphics;
  Quit;
end.
```

```
{ NOTE: After Compulation, rename EXE file to STEREO.EXE
}

{
============================================================
    GOSTEREO, Sterolithography Device Driver Front End
    for 3D Systems' Sterolithography System
================================================

  Recent History:
   4/20/88  Ver 3.10  Initial Release
   4/25/88  Ver 3.11  Correct command line pasing
}

{$R-,S-,I-,D-,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,0 } { No Heap Allocation }

Uses
  Dos;

  const
    DosName = 'C:\COMMAND.COM';
    ProgramName = '/C STEREONC';
    ProgramCOPName = '/C STEREOC';

  var
    regs: Registers;
    COP: Boolean;
    CommandLine,ProgramStr : String;
    i: Integer;

  begin
    { Get Parameters }
    CommandLine:='';
    for i:=1 to ParamCount do CommandLine := CommandLine
+ ParamStr(i);
```

```
  { Make Equipment check call }
  Intr($11,regs);
  COP := (regs.AL and 2)<>0;

  { Select the correct file }
  if COP then ProgramStr := ProgramCOPName
  else ProgramStr := ProgramName;
  Exec(DosName,ProgramStr+' '+CommandLine);
  Halt(DosExitCode);
end.
```

```
{$R-,S+,I+,D+,T-,F-,V+,B-,N-,L+ }
{$M 4096,0,0 }
{ SLA-1 Release 3.0X Installation Program  }

{*********

  4/00/88 ver1    r306  written this date
  7/18/88 ver2    r306  modified to r320
  7/21/88 ver21   correct NMI disable from port[305] to
port[$305]
  7/22/88 ver320  deleted writlns on super.prm
  7/25/88 ver320  add rmdir CAL in erase everything
  7/28/88 ver320  add option to format a diskette before
archive
  8/22/23 ver321  add ipconfig ipcust.sys hostname
process
  8/22/23 ver321  rename autoexec and config to
autoexec.sav, config.sav
  8/25/88 ver322  add stereo.def to update, add
installation ver #
 ********************************** end NOTES }

Program Install;
Uses
    Crt, Dos;
Const
     UpBytes = 1024000;
     NewBytes = 20000000;
Var
   CurDir      : String[80];
   BytesFree   : Longint;
   InChar, key : Char;
   regs        : registers;
   fn, fa, fc  : text;
   fileexists  : boolean;
   month,day,year,dayofweek : word;
```

```
Procedure DosCmnd(Command: string);
Begin
     writeln(Command);
     Exec('c:\command.com', Concat('/c ',Command));
     If ( DosError <> 0 ) then Halt;
     writeln;
End;

Procedure DosCmndNoHalt(Command: string);
Begin
     writeln(Command);
     Exec('c:\command.com', Concat('/c ',Command));
     If ( DosError <> 0 ) then writeln('   DOS return
code=',DosError);
     writeln;
End;

procedure ShowTop;
begin
   window(1,1,80,25);
   ClrScr;
   GotoXY(1,2);
   writeln('        Release 3.20  Stereolithography
Installation Program  ');
   writeln('                      Installation Version
3.22               ');
   Window(1,5,80,25);
end;

procedure UserWait;
begin
   writeln;
   writeln;
   writeln('              ... press any key when ready
to continue     ');
   repeat Delay(200) until Keypressed;
   key := Readkey;
```

```
   writeln;
end;

procedure ForceReBoot;
begin
     window(1,1,80,25);
     ClrScr;
     writeln;
     DosCmnd('type c:\3DSYS\read.me');
     writeln('               ... press any key when ready
to continue     ');
     repeat Delay(200) until Keypressed;
     key := Readkey;
     ShowTop;
     writeln;
     Writeln('            SLA R3.20 Installation
Completed.      ');
     writeln;
     writeln;
     writeln;
     writeln;
     writeln;
     writeln('      *****  PLEASE REMOVE THE FLOPPY DISK
IN DRIVE "A".    ');
     writeln;
     writeln;
     writeln('        When complete, press any key to
run the new ');
     writeln('          release 3.20 Stereolithography
Software :  ');
     writeln;
     UserWait;
   { Intr($19,regs); }
     port[$305] := port[$305] or 7;
     Inline($B8/$40/$00/$8E/$D8/$BB/$72/$00/$C7/$07/$34/
$00/$C7/$47/$01/$12/$00/
          $EA/$00/$00/$FF/$FF/
```

```
          $01/$04/$43/$4F/$50/$59/$03/$61/$26/$05/
$50/$41);
end;

procedure RecordDate;
begin
     ShowTop;
     {$I-}
     DosCmndNoHalt('mkdir c:\HISTORY');
     assign(fn,'c:\HISTORY\Maint.log');
     reset(fn);
     fileexists := (IOResult = 0);
     if not fileexists then rewrite(fn);
     append(fn);
     GetDate(Year,month,day,dayofweek);
     writeln(fn);
     if fileexists then write(fn,'   Added to this file:
')
                   else write(fn,'   Started this file:
');
     writeln(fn,month,'/',day,'/',year);
     write(fn,'   ACTION:');
     Case InChar of
       '1' : writeln(fn,' Installed new R3.20 SLA
software.');
       '2' : writeln(fn,' Updated R3.06 to R3.20 SLA
software.');
     end;
     close(fn);
end; { record date }

procedure ReLoadR320diskette;
begin
     ShowTop;
     writeln('   ***** Please reload the original R320
SLA installation diskette:');
     writeln('   ( SLA 3.20    Volume 1 of 1 )');
```

```
     UserWait;
end;

procedure ArchR306;
begin
     ShowTop;
     writeln('   ***** Please insert a high density
diskette for archiving R3.06:');
     writeln;
     writeln('      ... press  B  to bypass the
archive');
     writeln('             or  Q  to quit the program');
     writeln('             or  F  to format a diskette
and receive archive:');
     writeln('             or  A  when a previously
formated diskette ');
     writeln('                    is loaded and ready to
receive archive: ');
     repeat delay(100) until KeyPressed;
     key := upcase(readkey);
     if key = 'B' then begin
        writeln;
        writeln;
        writeln('  This is your last chance to archive
R3.06...   ');
        writeln('     ARE YOU SURE YOU WANT TO SKIP THE
ARCHIEVE ?');
        writeln;
        repeat delay(100) until KeyPressed;
        key := upcase(ReadKey);
        if key='Y' then key:='B' else key:='A';
     end;
     if  key =  'F' then DosCmnd('format a:');
     if  key =  'Q' then Halt;
     if  key <> 'B' then begin
        DosCmndNoHalt('mkdir a:\R306SLA');
        DosCmnd('copy c:\3DSYS\*.* a:\R306SLA');
```

```
        ReLoadR320diskette;
    end;
    writeln;
    DosCmnd('erase c:\3DSYS\*.exe');
end;

Procedure CopyQDOS;
begin
    ShowTop;
    writeln('   ***** Please load the QDos diskette:');
    writeln('   ( Q-DOSII by Gazelle Systems )');
    writeln;
    UserWait;
    { QDSTART makes its own QDOS directory. }
    DosCmnd('a:QDSTART');
    writeln;
    writeln;
    writeln('   QDOS II successfully loaded onto
harddisk.');
end;

procedure CopyFTP;
begin
    ShowTop;
    writeln;
    writeln('   ***** Please load the FTP diskette
10:');
    writeln('   ( MICOM- Interlan, Inc.  VOL X of X
)');
    UserWait;
    DosCmnd('copy a:\FTPSRV.exe c:\FTP');
    ShowTop;
    writeln;
    writeln('   ***** Please load the FTP diskette
6:');
    writeln('   ( MICOM- Interlan, Inc.  VOL VI of X
)');
```

```
      UserWait;
      writeln;
      DosCmnd('copy a:\FTP.exe c:\FTP');
      DosCmnd('copy a:\IFCONFIG.exe c:\FTP');
      DosCmnd('copy a:\IPCONFIG.exe c:\FTP');
      DosCmnd('copy a:\IPCUST.sys c:\FTP');
      DosCmnd('copy a:\NI5210.sys c:\FTP');
      DosCmnd('copy a:\TNVT100.exe c:\FTP\TELNET.exe');
      DosCmnd('\FTP\iPconfig \FTP\iPcust.sys hosttable
c:\ftp\host.txt');
      DosCmnd('\FTP\iPconfig \FTP\iPcust.sys bs');

      DosCmnd('\FTP\iFconfig \FTP\NI5210.sys 89.0.3.1');
      DosCmnd('\FTP\iFconfig \FTP\NI5210.sys Memory
D800');
      DosCmnd('ipconfig ipcust.sys hostname Process');
      writeln;
      writeln('   FTP successfully loaded onto
harddisk.');
end;

procedure CopyCAL;
begin
      DosCmndNoHalt('mkdir c:\CAL');
      DosCmndNoHalt('copy c:\3DSYS\*.cal c:\CAL\*.cal');
      DosCmndNoHalt('copy c:\WORK\*.cal c:\CAL\*.cal');
      DosCmndNoHalt('copy a:\CAL\*.geo c:\CAL\*.geo'); {
STEREO.GEO }
end;

procedure CopyR320Root;
begin
      ShowTop;
      Assign (fa,'c:\autoexec.bat');
      fileexists := (IOResult = 0);
      if fileexists then Rename (fa,'autoexec.sav');
      Assign (fc,'c:\config.sys');
```

```
      fileexists := (IOResult = 0);
      if fileexists then Rename (fc,'config.sav');
      DosCmnd('copy a:\Rootx\*.* c:\');
end;

procedure CopyR320Updates;
begin
   ShowTop;
   DosCmnd('copy a:\3DSYS\*.exe c:\3DSYS');
   DosCmnd('copy a:\3DSYS\stereo.def c:\3DSYS');
   DosCmnd('copy a:\3DSYS\read.me c:\3DSYS');
   DosCmnd('copy a:\rootx\chrono.sys c:\');
   writeln;
   CopyCAL;
   writeln;
end;

procedure CopyR320New;
begin
      ShowTop;
      DosCmnd('mkdir c:\3DSYS');
      writeln;
      DosCmnd('copy a:\3DSYS\*.* c:\3DSYS');
      DosCmnd('copy a:\3DSYS\*.pr^ c:\3DSYS\*.prm');
      DosCmnd('mkdir c:\3DSYS\EDITOR');
      DosCmnd('copy a:\3DSYS\Editor\*.*
c:\3DSYS\EDITOR');
      DosCmnd('mkdir c:\WORK');
      DosCmnd('copy a:\WORK\Sample.Mat c:\WORK');
      DosCmnd('mkdir c:\HISTORY');
      DosCmnd('mkdir c:\FIELD');
      DosCmnd('copy a:\Field\*.* c:\Field');
      DosCmnd('mkdir c:\FTP');
      DosCmnd('mkdir c:\CAL');
      DosCmnd('copy a:\FTP\Host.txt c:\FTP');
      writeln;
      writeln;
```

```
     writeln('   Primary SLA system successfully loaded
onto harddisk.')
end;

procedure ExplainUpdateR306toR320;
begin
   ShowTop;
   writeln;
   writeln('                Update release 3.06 to
release 3.20  ');
   writeln('                     INFORMATION PANEL' );
   writeln;
   writeln;
   writeln('   The first step in the update process is
to archive the release 3.20');
   writeln(' files residing in the \3DSYS directory. ');
   writeln;
   writeln('   Next, the SLA programs in the \3DSYS
directory are updated to the ');
   Writeln('   release 3.20 level.');
   writeln;
   writeln('   The .EXE files in \3DSYS will be deleted.
None of the .PRM files in');
   Writeln('   the \3DSYS or \WORK directory will be
modified.');
   Writeln;
   UserWait;
end;

procedure NewR320SLA;
{*** Perhaps this procedure should verify that no SLA
software exists.***}
begin
     CopyR320Root;
     CopyR320New;
     CopyQDOS;
     CopyFTP;
```

```
        RecordDate;
        ForceReBoot;
    end;

    Procedure UpdateR306toR320;
    Begin
        ShowTop;
        If ( UpBytes = BytesFree ) Then
        Begin
            writeln('******************************');
            writeln;
            Writeln('        The SLA R320 software update
requires ', UpBytes, ' Bytes');
            Writeln('             but only ', BytesFree, '
are currently available.');
            writeln;
            writeln;
            writeln('          Please clear enough memory
space on the hard disk');
            writeln('                and re-try the SLA
software update. ');
            writeln;
            writeln;
            writeln;
            Writeln('         Execution Terminated. ');
            Halt;
        End;
        ExplainUpdateR306toR320;
        ArchR306;
        CopyR320Updates;
        RecordDate;
        ForceReBoot;
    End;

    procedure ShowInstallMenu;
    begin
```

```
     Window(1,1,80,25);
     ClrScr;
     gotoxy(9,8);
     Writeln('*******  SLA-1 Installation Program
Primary Menu *******');
     Writeln;
     Writeln('                    1) Install a new R320
SLA System');
     Writeln('                    2) Update R306 to R320
SLA Software');
     Writeln('                    3) Information Panel ');
     Writeln;
     Writeln('                    Q) Quit Installation
Program');
     Writeln;
     Write  ('                         Enter Option:');
end;

procedure GiveHelpInfo;
begin
   ShowTop;
   writeln;
   writeln('                *****  Information Panel
*****');
   writeln;
   writeln;
   writeln('     This software allows fast, easy
installation of 3D System''s');
   writeln('     control computer environment. ');
   writeln;
   writeln('     A new installation of R3.20 requires a
QDOS II installation disk');
   writeln('     and the MICOM Interlan (FTP)
installation disks.');
   writeln;
   writeln('     For an update from R3.06 to R3.20 use a
formatted high density');
```

```
   writeln('     floppy disk for archiving the R3.06
software.  This is not mandatory,');
   writeln('     only suggested. ');
   writeln;
   writeln('     All customized parameter files (*.PRM)
should be kept in the \WORK');
   writeln('     directory.  ');
   writeln;
   writeln('     If you wish to install a color monitor
please contact 3D systems.');
   UserWait;
end;

procedure EraseEverything;
begin
   ShowTop;
   writeln('       ==============>  CRITICAL WARNING
<==============');
   writeln;
   writeln('    THIS IS OPTION ERASE EVERYTHING:     ');
   writeln;
   writeln;
   writeln;
   writeln(' ... press  E  to erase everything except');
   writeln('      the DOS directory and COMMAND.COM ');
   writeln;
   writeln('press any other key to leave this option ');
   writeln;
   repeat delay(100) until KeyPressed;
   key := upcase(readkey);
   if  key <> 'E' then exit;
   writeln;
   DosCmndNoHalt('c:');
   DosCmndNoHalt('erase c:\3DSYS\EDITOR\*.*');
   DosCmndNoHalt('cd c:\3DSYS');
   DosCmndNoHalt('rmdir EDITOR');
```

```
    DosCmndNoHalt('erase c:\3DSYS\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir 3DSYS');

    DosCmndNoHalt('erase c:\FIELD\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir FIELD');

    DosCmndNoHalt('erase c:\WORK\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir WORK');

    DosCmndNoHalt('erase c:\HISTORY\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir HISTORY');

    DosCmndNoHalt('erase c:\QDOS\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir QDOS');

    DosCmndNoHalt('erase c:\FTP\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir FTP');

    DosCmndNoHalt('erase c:\CAL\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir CAL');

    DosCmndNoHalt('erase c:\*.sys');
    DosCmndNoHalt('erase c:\*.bat');
    writeln;
    writeln('   Everything is erased.   ');
    writeln;
    UserWait;
end;

Begin
```

```
      ClrScr;
      BytesFree := DiskFree(3);
      DOScmnd('verify on');
      repeat
         ShowInstallMenu;
         InChar := 'Z';
         repeat
            repeat delay(100) until Keypressed;
            InChar:= upcase(ReadKey);
         until InChar in ['1','2','3','Q','Z'];
         Case InChar of
           '1': NewR320SLA;       { Install everything
from ground zero. }
           '2': UpdateR306toR320;  { This option archives
the old version
                      software and doesn't need qdos, et
al., installed. }
           '3': GiveHelpInfo;     { Passive help file. }
           'Q': exit;
           'Z': EraseEverything
         end;
       until false;
End.
```

^Z

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================

   MENU.PAS
   SLA-1 Menu Software

---------------------------------------------------------

   History:
   11/12/87  Ver 2.30  first Beta software release
   11/20/87  Ver 2.31  ENTER for Exit option fix
   11/23/87  Ver 2.32  pauses for Turbo Basic error
messages
   11/24/87  Ver 2.33  converted from Turbo Basic to
Turbo Pascal
                       SHELL's no longer eat up 64K or
main memory
                       Consolidated Profiler &
Calibrator
                       Added Materials Manager to
Utility Menu
                       Added Change Directory to Dos
Menu
                       Alt-F9 View Option History added
   12/02/87  Ver 2.34  fixed date format
   12/06/87  Ver 2.35  Using BEAM instead of CALIB and
PROFILE
   12/12/87  Ver 2.40  resets Text Mode when returning
from Shell
   12/16/87  Ver 2.41  clearing of screen for pause
fixed
                       prevent double pause (esp for
directory)
   12/17/87  Ver 2.42  revitalize cursor before running
Shells
   12/24/87  Ver 2.43  better cursor on/off routines
```

```
                        'X' input at ExitSystem verify
ignored
    1/08/88  Ver 2.50  TESTPART program access added
    1/25/88  Ver 2.60  version update only
    2/12/88  Ver 2.61  1900 + year in resetting system
date
    3/09/88  Ver 2.80  added copyright notice
                       installed new chronometer system
    4/04/88  Ver 3.00  added menu changes
    4/08/88  Ver 3.01  SUPER renamed to BUILD
    4/27/88  Ver 3.10  CHRONO.SYS file format changed
    5/09/88  Ver 3.11  modified to stop loop caused by
bad chrono.sys read
    6/10/88  Ver 3.12  add warning no on time scheduled
before entering build
    9/16/88  Ver 3.26  updated VersionId (must be
highest of all sw)
------------------------------------------------- }

{$R-,S-,I-,D+,T-,F-,V+,B-,N- }
{$M 4096,0,0 }

program MenuProgram;
uses Crt,Dos;

type
  str4   = string[4];
  str10  = string[10];
  str16  = string[16];
  str40  = string[40];
  str80  = string[80];
  str100 = string[100];
  HistoryType = record
    Menu,Option: Byte;
    mon,day,hour,min: Byte;
  end;
  DateType = record
```

```
    year : Word;
    month: Word;
    day  : Word;
  end;
  TimeStampType = record
    year : Word;
    month: Word;
    day  : Word;
    hour : Word;
    minu : Word;
    sec  : Word;
  end;
  YearType = (NotLeapYear,LeapYear);

var
  ActiveCurrentLaserOnEntryExists, QuickShow : boolean;
  Fnkey,CmdChr,key,key2: Char;
  IOerr,CmdVal,ordkey: Integer;
  Row,Menu,Indent: Integer;
  Mon,Day,Year,Hour,Minu,Sec,WeekDay: Integer;
  WhenToExecute,ChronoOffset: array[1..32] of LongInt;
  WhatToExecute: array[1..32] of str40;
  code,i,ChronoPtr: Integer;
  PausedFlag,ErrorFlag,NoInterval: Boolean;
  Line,CurrentDate,CurrentTime: str80;
  temp,SingleBar,DoubleBar: str100;
  CurrentDir: str40;
  cmd: str16;
  regs: Registers;
  SystemFile,ChronoFile,ErrorFile: Text;
  MenuPos: array[1..3] of Integer;
  OptionHistory: array[1..19] of HistoryType;
  StartHistory,EndHistory: Integer;
  MinTimeStamp: TimeStampType;
  CurrentMinutes: LongInt;
  UpdateFile: File of Char;
```

```
const
  MSDOS = $21;
  VersionId = '3.26';
  ComSpec = 'C:\COMMAND.COM';
  SystemFileName = 'C:\SLA1.SYS';
  ErrorFileName = 'C:\SYSTEM.ERR';
  ChronoFileName = 'C:\CHRONO.SYS';
  sp80 = '                                        ';
  DayOfWeek: String[21] = 'SunMonTueWedThuFriSat';
  Zel: array[1..12] of Integer =
(1,4,3,6,1,4,6,2,5,0,3,5);
  MainExecs: array[1..8] of str10 =
    ('XFER','MERGE','VIEW','PREPARE','BUILD','','','');
  UtilityExecs: array[1..8] of str10 =
    ('POWER','BEAM','ZSTAGE','MATERIAL','EDIT',
'TESTPART','','');
  DosExecs: array[1..8] of str10 =
    ('CHKDSK','','DIR/W','QD2','','','','');
  MenuMax: array[1..3] of Integer = (8,7,6);
  MenuColor: array[1..3] of Integer = (12,11,10);
  MaxHistory = 19;
  Cr = ^M;
  DaysInMonth: array[NotLeapYear..LeapYear,1..12] of
Word
    = ((31,28,31,30,31,30,31,31,30,31,30,31),
       (31,29,31,30,31,30,31,31,30,31,30,31));
  DayTable: array[NotLeapYear..LeapYear,1..13] of Word
    = ((0,31,59,90,120,151,181,212,243,273,304,334,365),

       (0,31,60,91,121,152,182,213,244,274,305,
335,366));

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
```

```
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure ResetScreen;
  begin
    TextColor(White);
    TextBackground(Black);
    TextMode(C80);
    CursorOff;
  end;

function upper(text: String): String;
var
  i: Integer;
  temp: String;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;
```

```
function many(Ch: Char;num: Integer): str80;
var
  temp: str80;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function spc(len: Integer): String;
begin
  spc := copy(sp80,1,len);
end;

procedure beep;
begin
  sound(1000);
  delay(100);
  NoSound;
end;

function IntStr(num: Integer): str10;
var
  temp: str10;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

procedure ReadCommandLine(var line: str80);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
```

```
    line := line + ' ' + ParamStr(i);
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Compute day-of-week using Harry's modified Zeller
Algorithm
}
function Zeller(month,day,year: Integer): Integer;
begin
  if month<3 then dec(year);
  Zeller := (Zel[month] + day + year + (year shr 2)) mod
7;
end;
```

```
{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetChipTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;    { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;    { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;    { get month }
              Mon := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $09;    { get year }
                Year := port[$71];
```

```
                port[$70] := $0A;
                if (port[$71] and $80)=0 then good :=
true;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
  Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
  WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetChipTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetChipTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
```

```
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

{
  Reset Computer System's Date and Time from Hardware
Clock
}
procedure SetSystemDateTime;
begin
  GetChipTime;
  SetDate(1900+Year,Mon,Day);
  SetTime(Hour,Minu,Sec,0);
end;

{
  Julian-Cummulative-Minute algorithms
}
function Julian(date: DateType): Word;
var
  YearLookup: YearType;
begin
  with date do begin
    if (year mod 4) = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
```

```
    Julian := 365*(year-1900) +
DayTable[YearLookup,month] +
      day + ((year-1901) div 4) - 32142 {1/1/88};
  end;
end;

procedure UnJulian(JulNum: Word;var date: DateType);
var
  YearLookup: YearType;
  offset,j,lastj: LongInt;
begin
  with date do begin
    inc(JulNum,32142);   {1/1/88}
    year := 1900;
    j := 0;
    lastj := 0;
    while j <= JulNum do begin
      lastj := j;
      if year mod 4 = 0 then inc(j,366)
        else inc(j,365);
      inc(year);
    end;
    dec(year);   { back off a year }
    if year mod 4 = 0 then YearLookup := LeapYear
      else YearLookup := NotLeapYear;
    offset := JulNum - lastj;
    month := 1;
    j := 0;
    lastj := 0;
    while j <= offset do begin
      lastj := j;
      j := DayTable[YearLookup,month];
      inc(month);
    end;
    dec(month,2);
    day := offset - lastj + 1;
  end;
```

```
end;

function AbsMinutes(TimeStamp: TimeStampType): LongInt;
var
  date: DateType;
begin
  with TimeStamp do begin
    date.year  := year;
    date.month := month;
    date.day   := day;
    AbsMinutes := LongInt(Julian(date)) * 1440 + (hour *
60 + minu);
  end;
end;

procedure AbsTimeStamp(Minutes: LongInt;var TimeStamp:
TimeStampType);
var
  date: DateType;
  DayMinutes: Word;
begin
  DayMinutes := Minutes mod 1440;
  UnJulian(Minutes div 1440,date);
  with TimeStamp do begin
    year  := date.year;
    month := date.month;
    day   := date.day;
    hour  := DayMinutes div 60;
    minu  := DayMinutes mod 60;
  end;
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
var
```

```
  s,e: Integer;
begin
  s := 1;
  while (Text[s] in ['"',' ']) and (s < length(Text)) do
inc(s);
  e := length(Text);
  while (Text[e] in ['"',' ']) and (e > s) do dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

function InChildProcess: Boolean;
begin
  InChildProcess := (Mem[0:99 shl 2] and $80)>0;
end;

procedure SetChildProcess;
begin
  Mem[0:99 shl 2] := $80;
end;

procedure ClearChildProcess;
begin
  Mem[0:99 shl 2] := $00;
end;

procedure CheckForOptions(var line: str80);
var
  i: Integer;
begin
  line := upper(line);
  i := pos('/NOINTERVAL',line);  { check for /NOINTERVAL
}
  if i>0 then begin
    NoInterval := true;
    delete(line,i,11);
  end;
end;
```

```
procedure Pause;
begin
  if not PausedFlag then begin
    gotoxy(1,25);
    write('Press Enter to Continue... ');
    readln(Input);
    PausedFlag := true;
  end;
end;

function Desc(menu,line: Integer): str80;
var
  index: Integer;
begin
  index := menu * 10 + line;
  case index of
    11: Desc := 'Transfer Files Between this Computer
and Another over Ethernet';
    12: Desc := 'Merge Separately-Sliced Object Files
Together';
    13: Desc := 'Graph an Object in either Faceted or
Sliced form';
    14: Desc := 'Prepare to Make a Part and Set Up
Various Parameters';
    15: Desc := 'Run the Stereolithography Part-Builder
Program';
    16: Desc := 'Go to the Utility Menu';
    17: Desc := 'Go to the Disk Operating System Menu';
    18: Desc := 'Exit Stereolithography System
Completely';
    19: Desc := 'WARNING: No laser on time scheduled.
Contine anyway ?';
    21: Desc := 'Control the Power-Up or Down Sequence
of Laser Equipment';
    22: Desc := 'Laser Beam Profile and Calibration
Operations';
```

```
    23: Desc := 'Move the Elevator Up and Down';
    24: Desc := 'View, Alter, and Add Data for Vat
Materials';
    25: Desc := 'Enter the Editor to Modify the Contents
of a File';
    26: Desc := 'Run the Test Part Utility Program';
    27: Desc := 'Return to the Main Menu';
    31: Desc := 'Show Hard Disk Statistics, including
Free Space Remaining';
    32: Desc := 'Change to a Different File Directory';
    33: Desc := 'Show the Directory for the Current
Directory';
    34: Desc := 'Enter the Quick-DOS Utility to Move
Files and Change Directories';
    35: Desc := 'Enter a DOS Shell, Allowing You to Run
Other Programs';
    36: Desc := 'Return to the Main Menu';
    else Desc := '';
  end;
end;

{-------------------------------------------------------}

procedure DrawMain;
begin
  writeln('                          MAIN MENU');
  TextColor(White);
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  writeln;
  gotoxy(Indent,whereY);
  writeln('1. Network       ');
  gotoxy(Indent,whereY);
  writeln('2. Merge');
  gotoxy(Indent,whereY);
  writeln('3. View  ');
```

```
  gotoxy(Indent,whereY);
  writeln('4. Prepare');
  gotoxy(Indent,whereY);
  writeln('5. Build');
  gotoxy(Indent,whereY);
  writeln('6. Utility Menu');
  gotoxy(Indent,whereY);
  writeln('7. DOS Menu');
  gotoxy(Indent,whereY);
  writeln('X. Exit');
end;

procedure DrawUtility;
begin
  writeln('                          UTILITY MENU');
  TextColor(White);
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  writeln;
  gotoxy(Indent,whereY);
  writeln('1. Power Sequencer');
  gotoxy(Indent,whereY);
  writeln('2. Beam Analysis');
  gotoxy(Indent,whereY);
  writeln('3. Elevator Mover');
  gotoxy(Indent,whereY);
  writeln('4. Material Manager');
  gotoxy(Indent,whereY);
  writeln('5. Edit a File');
  gotoxy(Indent,whereY);
  writeln('6. Make Test Parts');
  gotoxy(Indent,whereY);
  writeln('X. Return to Main Menu');
end;

procedure DrawDos;
```

```
begin
  writeln('                                DOS MENU');
  TextColor(White);
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  writeln;
  gotoxy(Indent,whereY);
  writeln('1. Disk Usage Stats');
  gotoxy(Indent,whereY);
  writeln('2. Change Directory');
  gotoxy(Indent,whereY);
  writeln('3. Directory Listing');
  gotoxy(Indent,whereY);
  writeln('4. Quick DOS Utility');
  gotoxy(Indent,whereY);
  writeln('5. DOS Shell');
  gotoxy(Indent,whereY);
  writeln('X. Return to Main Menu');
end;

procedure DrawMenuScreen;
begin
  SetSystemDateTime;
  ClrScr;
  TextColor(White);
  writeln(DoubleBar);
  writeln(' SLA-1 3D SYSTEMS STEREOLITHOGRAPHY SYSTEM V'
    + VersionId);
  writeln('        All Programs and Screens Copyright
(C) 1988 by 3D Systems, Inc.');
  writeln(DoubleBar);
  writeln;
  TextColor(MenuColor[Menu]);
  case Menu of
    1: DrawMain;
    2: DrawUtility;
```

```
    3: DrawDos;
  end;
  writeln;
  gotoxy(20,whereY);
  writeln(copy(SingleBar,1,40));
  Row := whereY;
  writeln;
  writeln('        Select an Option either by Using the
Arrows and Pressing Enter');
  write  (' or by Pressing the Corresponding Number');
end;

{
  Show Menu Item Description
}
procedure ShowDescription;
begin
  gotoxy(Indent-2,9+MenuPos[Menu]);
  TextColor(Yellow);
  write(chr(16));
  gotoxy(1,24);
  write(spc(79));
  if quickShow then gotoxy((78-length(Desc(Menu,9))) div
2,24)
               else
gotoxy((78-length(Desc(Menu,MenuPos[Menu]))) div 2,24);
  TextColor(Black);
  TextBackground(MenuColor[Menu]);
  if quickShow then write(' ',Desc(Menu,9),' ')
           else write(' ',Desc(Menu,MenuPos[Menu]),' ');
  TextColor(White);
  TextBackground(Black);
end;

{
  chronometry routines
}
```

```
procedure GetCurrentMinutes;
begin
  MinTimeStamp.year  := 1900 + Year;
  MinTimeStamp.month := Mon;
  MinTimeStamp.day   := Day;
  MinTimeStamp.hour  := Hour;
  MinTimeStamp.minu  := Minu;
  CurrentMinutes := AbsMinutes(MinTimeStamp);
end;

function IntParse: Integer;
var
  num: Real;
  IntNum,code,code2: Integer;
begin
  if length(temp)=0 then IntNum := 0
  else begin
    while copy(temp,1,1) = ' ' do delete(temp,1,1);
    val(temp,num,code);
    if code<>0 then val(copy(temp,1,code-1),num,code2);
    IntNum := trunc(num);
    if code=0 then temp := ''
      else temp := copy(temp,code+1,99);
  end;
  IntParse := IntNum;
end;

procedure ReadChronoFile;
var
  ChronoLine: String;
  TimeStamp: TimeStampType;
  DayOffset,HourOffset,MinuOffset: Integer;
begin
  ActiveCurrentLaserOnEntryExists := false;
  ChronoPtr := 0;
  assign(ChronoFile,ChronoFileName);
  {$I-}
```

```
  reset(ChronoFile);
  {$I+}
  if IOresult<>0 then exit;
  while not eof(ChronoFile) do begin
    readln(ChronoFile,ChronoLine);
    if (ChronoLine <> '') and not (ChronoLine[1] in
['!','S']) then begin
      inc(ChronoPtr);
      temp := upper(copy(ChronoLine,6,255));
      with TimeStamp do begin
        month := IntParse;
        day   := IntParse;
        year  := 1900 + IntParse;
        hour  := IntParse;
        minu  := IntParse;
        if copy(temp,1,2) = 'AM' then begin
          if hour = 12 then hour := 0;
        end else begin   { PM }
          if hour < 12 then inc(hour,12);
        end;
        temp := copy(temp,4,255);
        WhenToExecute[ChronoPtr] := AbsMinutes
(TimeStamp);
      end;
      DayOffset  := IntParse;
      HourOffset := IntParse;
      MinuOffset := IntParse;
      ChronoOffset[ChronoPtr] := (LongInt(DayOffset) *
24 + HourOffset) * 60
        + MinuOffset;
      i := pos('!',temp);
      WhatToExecute[ChronoPtr] :=
NoQuotes(copy(temp,1,i-2));
      if pos('LASER ON NOW',WhatToExecute[ChronoPtr]) >
0 then begin
        GetCurrentMinutes;
```

```
          if (CurrentMinutes+31) <  WhenToExecute
[ChronoPtr]
          then  ActiveCurrentLaserOnEntryExists := true;
        end;
      end;
    end;
    close(ChronoFile);
end;

procedure UpdateChronoFile(index: Integer;NewTimeStr:
str16);
var
  i,j: Integer;
  ChronoLine: String;
  OldUpdateFilePos,UpdateFilePos: LongInt;
begin
  i := 0;
  assign(ChronoFile,ChronoFileName);
  {$I-}
  reset(ChronoFile);
  {$I+}
  if IOresult<>0 then exit;
  UpdateFilePos := 0;
  while not eof(ChronoFile) do begin
    OldUpdateFilePos := UpdateFilePos;
    readln(ChronoFile,ChronoLine);
    inc(UpdateFilePos,length(ChronoLine)+2);
    if (ChronoLine <> '') and not (ChronoLine[1] in
['!','S']) then begin
      inc(i);
      if i = index then begin
        close(ChronoFile);
        assign(UpdateFile,ChronoFileName);
        {$I-}
        reset(UpdateFile);
        {$I+}
        if IOresult<>0 then exit;
```

```
          Seek(UpdateFile,OldUpdateFilePos+6);
          for j:=1 to length(NewTimeStr) do
            write(UpdateFile,NewTimeStr[j]);
          close(UpdateFile);
          exit;
        end;
      end;
    end;
    close(ChronoFile);
end;

procedure SkipChronoFileEntry(index: Integer);
const
  CharVar: Char = 'S';
var
  i,j: Integer;
  ChronoLine: String;
  OldUpdateFilePos,UpdateFilePos: LongInt;
begin
  i := 0;
  assign(ChronoFile,ChronoFileName);
  {$I-}
  reset(ChronoFile);
  {$I+}
  if IOresult<>0 then exit;
  UpdateFilePos := 0;
  while not eof(ChronoFile) do begin
    OldUpdateFilePos := UpdateFilePos;
    readln(ChronoFile,ChronoLine);
    inc(UpdateFilePos,length(ChronoLine)+2);
    if (ChronoLine <> '') and not (ChronoLine[1] in
['!','S']) then begin
      inc(i);
      if i = index then begin
        close(ChronoFile);
        assign(UpdateFile,ChronoFileName);
        {$I-}
```

```
          reset(UpdateFile);
          {$I+}
          if IOresult<>0 then exit;
          Seek(UpdateFile,OldUpdateFilePos);
          write(UpdateFile,CharVar);
          close(UpdateFile);
          exit;
        end;
      end;
    end;
    close(ChronoFile);
end;

procedure ChronoExecute(index: Integer);
var
  NextTime: TimeStampType;
  tmon,tday,tyear,thour,tminu: str4;
begin
  gotoxy(1,24);
  ClrEol;
  TextColor(Black);
  TextBackground(White);
  gotoxy(25,24);
  write('   PLEASE WAIT MOMENTARILY   ');
  TextColor(White);
  TextBackground(Black);
  Exec(ComSpec,'/C '+WhatToExecute[Index]);
  while keypressed do key := ReadKey;   { flush out
keyboard buffer }
  ResetScreen;
  DrawMenuScreen;
  ShowDescription;
  if ChronoOffset[Index] > 0 then begin
    GetCurrentMinutes;
    WhenToExecute[Index] := CurrentMinutes +
ChronoOffset[Index];
    AbsTimeStamp(WhenToExecute[Index],NextTime);
```

```
    with NextTime do begin
      tmon  := PadSp(month);
      tday  := LZero(day);
      tyear := LZero(year-1900);
      thour := LZero(hour);
      tminu := LZero(minu);
      UpdateChronoFile(Index,tmon+'/'+tday+'/'+tyear+'
'+thour+':'+tminu);
    end;
  end else SkipChronoFileEntry(Index);
  ReadChronoFile;
end;

function ErrorOnScreen: Boolean;
var
  BaseAddr: Word;
  ScanError: Boolean;
  i: Integer;
begin
  if LastMode=7 then BaseAddr := $B000
    else BaseAddr := $B800;
  i := 0;
  ScanError := true;
  while (i<4000) and ScanError do begin
   if upcase(chr(Mem[BaseAddr:i]))='E' then
      if (upcase(chr(Mem[BaseAddr:i+2]))='R') and
        (upcase(chr(Mem[BaseAddr:i+4]))='R') then
ScanError := false;
    inc(i,2);
  end;
  ErrorOnScreen := not ScanError;
end;

{
  SHELL program, check for existence of error
  If error, pause and wait for Enter before returning to
menu
```

```
}
procedure Shell(Prog,Options: str80);
begin
  PausedFlag := false;
  SetSystemDateTime;
  CursorOn;
  Exec(Prog,Options);
  CursorOff;
  {$I-}
  assign(ErrorFile,ErrorFileName);
  reset(ErrorFile);
  {$I+}
  ErrorFlag := (IOresult=0);
  if ErrorFlag then begin
    close(ErrorFile);
    Pause;
    {$I-}
    erase(ErrorFile);
    {$I+}
  end else if ErrorOnScreen then Pause;
  ReadChronoFile;
end;

{-------------------------------------------------}

{
  Option History routines
}
procedure AddHistory(text: str10);
begin
  OptionHistory[EndHistory].Menu := Menu;
  OptionHistory[EndHistory].Option := CmdVal;
  OptionHistory[EndHistory].mon := Mon;
  OptionHistory[EndHistory].day := Day;
  OptionHistory[EndHistory].hour := Hour;
  OptionHistory[EndHistory].min := Minu;
  inc(EndHistory);
```

```
  if EndHistory > MaxHistory then EndHistory := 1;
  if StartHistory = EndHistory then begin
    inc(StartHistory);
    if StartHistory > MaxHistory then StartHistory := 1;
  end;
end;

procedure ShowHistory;
begin
  ClrScr;
  writeln('Menu System Recent Option History:');
  writeln;
  writeln('Menu  Option  Name            Date   Time');
  writeln('----  ------  --------------  -----  -----');

  i := StartHistory;
  while i<>EndHistory do with OptionHistory[i] do begin
    write('  ',Menu,'      ',Option,'    ');
    case Menu of
      1: cmd := MainExecs[Option];
      2: cmd := UtilityExecs[Option];
      3: cmd := DosExecs[Option];
    end;
    write(cmd,spc(16-length(cmd)));
    writeln(PadSp(mon),'/',LZero(day),'
',PadSp(hour),':',LZero(min));
    inc(i);
    if i > MaxHistory then i := 1;
  end;
  writeln;
  write('Press any key to continue... ');
  key := ReadKey;
  DrawMenuScreen;
end;

{
  Get Input for Menu, handle scrolling
```

```
}
procedure HelpAsk;
var
  i: Integer;
  LineChange,Leave: Boolean;
begin

  Leave := false;
  LineChange := true;
  repeat
    if LineChange then ShowDescription;
    repeat
      GetChipTimeStamp;
      GetDir(0,CurrentDir);
      gotoxy(2,5);
      write(CurrentDir);
      gotoxy(67,5);
      write(CurrentDate);
      gotoxy(71,6);
      write(CurrentTime);
      GetCurrentMinutes;
      gotoxy(1,1);
      for i:=1 to ChronoPtr do begin
        if WhenToExecute[i] <= CurrentMinutes then
ChronoExecute(i);
      end;
    until keypressed;

    LineChange := false;
    CmdChr := upcase(ReadKey);
    if CmdChr = chr(0) then FnKey := ReadKey
      else FnKey := chr(0);
    if CmdChr in [#0,'1'..'9','X',Cr] then begin
      LineChange := true;
      gotoxy(Indent-2,9+MenuPos[Menu]);
      write(' ');
```

```
      if (CmdChr=Cr) and (MenuPos[Menu]<MenuMax[Menu])
then
        CmdChr := chr(ord('0')+MenuPos[Menu]);
      if (CmdChr='X') or ((CmdChr=Cr) and
(MenuPos[Menu]=MenuMax[Menu])) then
        CmdChr := chr(ord('0')+MenuMax[Menu]);
      case FnKey of
        'P': if MenuPos[Menu] >= MenuMax[Menu] then
MenuPos[Menu] := 1
               else inc(MenuPos[Menu]);
        'H': if MenuPos[Menu] <= 1 then MenuPos[Menu] :=
MenuMax[Menu]
               else dec(MenuPos[Menu]);
        'p': ShowHistory;
      end;
      val(CmdChr,CmdVal,code);
      if (CmdVal>0) and (CmdVal<=MenuMax[Menu]) then
begin
        Leave := true;
        MenuPos[Menu] := CmdVal;
      end;
    end;
  until Leave;
  case Menu of
    1: cmd := MainExecs[CmdVal];
    2: cmd := UtilityExecs[CmdVal];
    3: cmd := DosExecs[CmdVal];
  end;
  if length(cmd)>0 then AddHistory(cmd);
end;

{---------------- UTILITY STUFF ------------------}

procedure UtilityMenu;
begin
  repeat
    Menu := 2;
```

```
    DrawMenuScreen;
    HelpAsk;
    ClrScr;
    if CmdVal<7 then begin
      Shell(ComSpec,'/C '+UtilityExecs[CmdVal]);
      ResetScreen;
    end;
  until CmdVal = MenuMax[2];
end;

{---------------- DOS STUFF ------------------}

procedure DiskUsage;
begin
  writeln('       DISK USAGE STATISTICS');
  writeln(copy(SingleBar,1,34));
  writeln;
  Shell(ComSpec,'/C '+DosExecs[1]);
  Pause;
end;

procedure DirList;
begin
  Shell(ComSpec,'/C '+DosExecs[3]);
  Pause;
end;

procedure ChangeDir;
var
  NewDir: str40;
begin
  CursorOn;
  ClrScr;
  writeln('Current directory is ',CurrentDir);
  writeln;
  writeln;
```

```
  writeln('You can specify a new directory using the
following rules:');
  writeln;
  writeln('If you wish to move to a subdirectory, just
type in the');
  writeln('subdirectory''s name.');
  writeln;
  writeln('If you want to move up a directory level,
just enter .. (two periods).');
  writeln;
  writeln('If you are going to specify a completely new
directory path, start');
  writeln('your entry with a \ (backslash).');
  writeln;
  writeln;
  write('Enter new directory specification: ');
  readln(NewDir);
  {$I-}
  ChDir(NewDir);
  if IOresult<>0 then begin
    writeln;
    writeln('Can''t find directory.');
    writeln;
    write('Press any key to continue... ');
    key := ReadKey;
  end;
  {$I+}
  CursorOff;
end;

procedure QDos;
begin
  Shell(ComSpec,'/C '+DosExecs[4]);
end;

procedure DosShell;
begin
```

```
  CursorOn;
  ClrScr;
  SetChildProcess;
  writeln('To return to the Dos Menu, use the EXIT
command.');
  writeln(copy(SingleBar,1,48));
  Exec(ComSpec,'');
  ResetScreen;
  ClearChildProcess;
end;

procedure DosMenu;
begin
  repeat
    Menu := 3;
    DrawMenuScreen;
    HelpAsk;
    ClrScr;
    case CmdVal of
      1: DiskUsage;
      2: ChangeDir;
      3: DirList;
      4: QDos;
      5: DosShell;
    end;
  until CmdVal = MenuMax[3];
end;

{------------------- MAIN STUFF ---------------------}

procedure ExitSystem;
begin
  CursorOn;
  ClrScr;
  writeln('Dos Menu Option 5 allows you to issue DOS
commands and run other');
```

```
  writeln('programs, then return to the Menu System
using the EXIT command.');
  writeln;
  writeln('This option will cause you to leave the
Stereolithography System');
  write  ('completely.  Are you sure you want to do
this? ');
  repeat
    key := upcase(ReadKey);
  until key<>'X';
  writeln(key);
  if key='Y' then begin
    SetSystemDateTime;
    Halt(0);
  end;
  CursorOff;
end;

procedure MainMenu;
  var
    mykey : char;
begin
  ResetScreen;
  repeat
    Menu := 1;
    DrawMenuScreen;
    HelpAsk;
    exec('C:\COMMAND.COM','/C SERVICE LASER OFF
+120/i/t');
    if CmdVal=5 then begin
       if ActiveCurrentLaserOnEntryExists then
mykey:='Y'
       else begin
          quickShow := true;
          ShowDescription;
          quickShow := false;
          Beep;
```

```
            repeat delay(100) until Keypressed;
            mykey:= upcase(ReadKey);
          end;
      end;
      ClrScr;
      if CmdVal=MenuMax[1] then ExitSystem
      else if CmdVal<6 then begin
        if (CmdVal<>5) or (mykey='Y') then begin
          Shell(ComSpec,'/C '+MainExecs[CmdVal]);
          ResetScreen;
        end;
      end else if CmdVal=6 then UtilityMenu
      else if CmdVal=7 then DosMenu;
    until 0=1;  { forever }
  end;

  {
    read SLA-1 System File
  }
  procedure ReadSystemFile;
  label
    NoMoreSystemFile;
  begin
    {$I-}
    assign(SystemFile,SystemFileName);
    reset(SystemFile);
    {$I+}
    if IOresult<>0 then begin
      writeln('No system file.  Stopping.');
      halt(1);
    end;
    repeat
      readln(SystemFile,Line);
    until eof(SystemFile) or ((length(Line)>0) and
  (Line[1]<>'!'));
    if not eof(SystemFile) then begin
      Line := NoQuotes(Line);
```

```
    CheckForOptions(Line);
    i := 1;
    while (i<2) and not eof(SystemFile) do begin
      Line := '';
      while (length(Line)=0) or (Line[1]='!') do begin
        if eof(SystemFile) then goto NoMoreSystemFile;
        readln(SystemFile,Line);
      end;
      Line := upper(Line);
      case i of
        1: begin
{ ????? }
           end;
      end;
      inc(i);
    end;
  end;
NoMoreSystemFile:
  close(SystemFile);
end;

{  -  -  -  -  -  }

begin
  QuickShow := false;
  CheckBreak := false;
  if InChildProcess then begin
    writeln('You are currently in a DOS Shell.');
    writeln('Please use EXIT to return to the Menu
System.');
    halt(2);
  end;
  Indent := (80-20) div 2;
  NoInterval := false;
  for i:=1 to 3 do
    MenuPos[i] := 1;
  StartHistory := 1;
```

```
    EndHistory := 1;

    SingleBar := many(#196,79);
    DoubleBar := many(#205,79);

    SetSystemDateTime;
    {$I-}
    assign(ErrorFile,ErrorFileName);
    erase(ErrorFile);
    IOerr := IOresult;
    {$I+}
    ReadSystemFile;
    ReadChronoFile;
    ReadCommandLine(Line);
    CheckForOptions(Line);
    MainMenu;
end.
^Z
```

```
{  To compile: use MKSTEREO Batch File  }

{
============================================================

    STEREO, Stereolithography Device Driver

    for 3D Systems' Stereolithography System

    Version 3.16  by Tarnz Technologies

============================================================
```

STEREO is a memory-resident laser controller device driver. It interfaces between the Background Task's Software Interrupt Handler and a program wanting to control the laser apparatus using easy text-based commands.

STEREO receives and processes simple (X,Y) laser position commands passed to it by some other program using a software interrupt vector. The CLIFF Turbo Pascal program is just such a program; it can read commands from either the computer's keyboard or from disk files and then pass them along to this program.

STEREO, like an MSDOS driver, can be called by programs written in any language, including assembly and BASIC. The interface will usually be a special subroutine that invokes the STERO software interrupt. Unlike an MSDOS driver, STEREO cannot be referenced directly by a filename.

STEREO's device driver format is simpler, faster, and more clean than an MSDOS device driver. STEREO processed a complete text string (a line) at a time, as

opposed to one character at a time for an MSDOS device driver. If desired, an official MSDOS device driver (STL:, LSR: or whatever) could be added in later.

STEREO understands the following commands:

	BX 0-65535	Beginning X position	(position)
	BY 0-65535	Beginning Y position	(position)
	JX 0-65535	Jump to new X position	(position)
	JY 0-65535	Jump to new Y position	(position)
	NX 0-65535	Next X position	(position)
	NY 0-65535	Next Y position	(position)
	SS 1-65535	Step Size	(rel incr)
!	SP 5-6553,..	multi-pass Step Periods	(10 us)
	SD 0-65535	Scanning Delay	(step periods)
	JP 5-6553	Jump Period	(10 us)
	JS 1-65535	Jump Size	(rel incr)
	JD 0-65535	Jump Delay	(step periods)
	LO 0-65535	Laser On delay	(10 us)
	LF 0-65535	Laser Off delay	(10 us)
+	NS	eNter Sequence mode	-
+	AB	Enter Absolute mode	-
	EX	Execute	-
	EC	Execute and Clear	-
	CL	Clear table	-
!	MD AS +/-	set Auto-Shutter mode	(on/off)
!	MD BL +/-	set BLock mode	(on/off)
!	MD PA +/-	set Pos Ack mode	(on/off)
!	SR	Soft Reset	
!	HR	Hard Reset	
!	RC 1-10	Redraw pass Count	(# passes)
!	RD 0-65535	Redraw Delay	(step periods)
!	RS 0-65535	Redraw Size	(distance)
!	VC 1-10	riVet pass Count	(# passes)
!	VP 5-6553,..	riVet step Periods	(10 us)
!	VR 0-65535	riVet Reduction amount	(distance)
!	VX 0-65535	riVet end X position	(position)

```
        ! VY 0-65535    riVet end Y position (position)
        ! WI 0-32767    Wait for Idle        (delay)

        - New Version 2.80 Commands -

        ! DC +/-        set Drift Correction mode (on/off)
        ! R1 0-65535,0-65535    new sensor #1 Reference
(X,Y) position
        ! R2 0-65535,0-65535    new sensor #2 Reference
(X,Y) position
        ! D1 0-65535,0-65535    new sensor #1 Drift
adjustment (X,Y) position
        ! D2 0-65535,0-65535    new sensor #2 Drift
adjustment (X,Y) position
        ! BG 0-65535,0-65535    define BeGin (X,Y)
position
        ! JM 0-65535,0-65535    JuMp to new (X,Y) position
                                ( preferred over JX, JY
combination )
        ! MV 0-65535,0-65535    MoVe to next (X,Y)
position
                                ( preferred over NX, NY
combination )
        ! RV 0-65535,0-65535    make a RiVet to next (X,Y)
position
                                ( preferred over VX, VY
combination )

    where:

      10 us         is the # of 10s of microseconds
                    resolution = 0.84 us (from 1.19 MHz
clock)
      position      is the absolute position in the field
      distance      is the absolute distance in step size
units
```

rel incr is the relative # of position increments
passes is the number of redraw passes
step periods is the time to cover that # of steps programmed by SP command (= tick counts)
delay delay value in milliseconds (may be faster on faster computers)

In the above list, following each two-character command is the range of numbers allowed as arguments to the command. Each command must be on a separate line, ending with a CR or CR/LF sequence. All spaces are ignored. All numbers are in unsigned decimal.

The Control Instruction Set is intended to be compatible with General Scanning's DG Series Electronic Packages for Laser Control. Only the Vector Mode is supported. The Raster Mode is not supported. The step and delta functions of the Vector Mode are also unsupported. No Z-axis control is supported yet. Commands marked with an '*' are not fully implemented as per the DG specification for functional reasons. Commands marked with '+' are included only for the sake of complete command compatibility. '!' denotes new commands not available on the DG devices.

DG Version 2.6 Vector Mode Commands not supported:

ST	enter STep mode
RE	REset to top of stack in step mode
DL	enter DeLta mode
Q1	Query IB 1
Q2	Query IB 2

The command line with invokes STEREO and causes it to be installed in memory takes the form:

```
STEREO [/options]
```

where /options are any number of user-selectable options:

```
/D    - debug mode, display vector information
/NV   - no constant vector velocity
/NG   - no geometric correction
/NC   - no version check at install time
```

STEREO can only be installed once after the computer has been powered up. If different options are required, the computer system will have to be completely rebooted.

The SP (and VP) command has multiple arguments. Each of up to ten passes of redraw can have its own step period. The SP command can have 1 to 10 arguments, each a step period for a pass:

```
SP sp0,sp1,sp2,sp3...
```

where sp0 is the step period for redraw pass 0 (the original vector move), sp1 is the step period for redraw pass 1, and so on. /V121 option is not longer supported.

Geometric correction is automatically invoked. If geometric correction is not desired, the option /NG or /GEO NO should be specified on the command line. Geometric correction occurs on the endpoints of any vector STEREO passes to the software interrupt handler.

STEREO code came from Version 1.40 of CLIFF, which originally performed the command translation. This

memory-resident STEREO device driver was formed to speed up 3D Systems' supervisory programs, written in GWBASIC and Turbo BASIC. With the Geo Correct Table, STEREO takes up approximately 80K of memory, increasing to about 150K the amount of main memory used by the complete laser controller system and its data structures.

Axes Control Options:

DEFAULT

/X +/-	select X-axis direction	+
/Y +/-	select Y-axis direction	+
/SWAPXY	swap X and Y coordinates	no
/GEO XY|YX|NO	select geo correct sequence (used for old style)	YX
/AXESBEGIN	to axes-control begin pos	no

==========

Recent History:

8/18/87	Ver 2.02	STEREO released memory resident portion of Ver 1.x CLIFF process WI Wait for Idle command added
8/22/87	Ver 2.03	Stuart's new Geometric Correction Y is the first, X is the second mirror /NC no version check flag jump vectors of 0 steps skipped CvtInt now works for 32768 ($8000) Step Periods supported for stream (non-block) mode
8/25/87	Ver 2.04	Geometric Correction delta bugs fixed

```
 9/23/87  Ver 2.12  STEREO Axes Control added
                    new options /X, /Y, /SWAPXY, /GEO
 9/30/87  Ver 2.25  Table-Interpolation Geometric
                    Correction installed
10/26/87  Ver 2.26  Wait for Idle argument is 100
                    minimum
                    /NG prevents loading of STEREO.GEO
10/28/87  Ver 2.27  improved Geometric Correction
                    formula
11/12/87  Ver 2.30  first Beta software release
11/18/87  Ver 2.31  allow axes control independent of
                    geo correction
                    /AXESBEGIN option added to
                    axes-control BX,BY
                    GeoCorrectTable size made
                    programmable (70 elements/axis)
                    added TableDivisor for
                    GeoCorrectTable
12/11/87  Ver 2.40  STEREO upgraded to Turbo Pascal
Version 4.0
                    STEREO always returns with Queue
                    Size Change in CX
                    GeoCorrectTable storage allocated
                    on heap dynamically
                    max GeoCorrectTable size increased
                    to 90 by 90
                    Some variables changed to Word or
                    LongInt type
12/16/87  Ver 2.41  GeoCorrect overwrite / memory
                    alloc debugged
                    QueueCount substituted for
                    QueueSize calculation
12/21/87  Ver 2.43  Check for Break turned off
                    improved keyboard and Ctrl-Break
                    Abort service
```

```
1/06/88  Ver 2.50  zero-length vectors ignored
                   Ctrl-Break check looks at scan
                   code now
1/27/88  Ver 2.60  better STEREO.GEO processing
                   blanks allowed around commas
                   separating numbers
                   check for too few or too many
                   entries in STEREO.GEO
                   no mark and redraw around executes
                   when redraw count is 1
                   OkayForRedrawCommand flag added to
                   limit redundant
                   mark and redraw commands
3/09/88  Ver 2.80  added copyright notice
                   implemented drift correction logic
                   for profiler sensors
                   reduced ASCII overhead for begin,
                   jump, and next commands
                   installed support for binary
                   STEREO commands
                   new STEREO commands:
                   DC, R1, R2, D1, D2, BG, JP, MV, RV
4/09/88  Ver 3.00  production release
4/18/88  Ver 3.01  fixed internal version number
                   drift correction defaults to on
4/18/88  Ver 3.10  Support Numeric Coprocessor
4/27/88  Ver 3.11  STEREO memory resident code size
                   increased
5/02/88  Ver 3.12  Drift Correction Y-axis bug
                   removed
5/13/88  Ver 3.13  JP renamed to JM, JP now reserved
                   Jump Period
5/16/88  Ver 3.14  new multi-timebase LASER support
6/06/88  Ver 3.15  improved Jump technique using
                   multi-timebase
7/01/88  Ver 3.16  correct Laser Unit for Co
                   Processor
```

```
                        check for Multi Timebase afer
                        checking for LASER
    7/06/88  Ver 3.20   debugged Numeric Coprocessor
                        uses alternate LibTrunc and
                        LibRound routines
    7/18/88  Ver 3.20   update MinSystemVersion
}

  {$R-,S-,I-,D-,T-,F-,V-,B-,L- }
  {$M 16384,0,65536 }

  { CoProcessor Support }
  {$IFOPT N+}
    {$DEFINE COP}
  {$ELSE}
    {$UNDEF COP}
  {$ENDIF}

uses Dos,
     Crt,
     Laser,
     LibNC;

type
  {$IFDEF COP}
  FloatType = Single;
  {$ELSE}
  FloatType = Real;
  {$ENDIF}
  str4   = string[4];
  str16 = string[16];
  str80 = string[80];
  BitArray = array[0..15] of Word;
  GeoCorrectTableType = array[0..10000] of LongInt;
  GeoCorrectTablePtr  = ^GeoCorrectTableType;
  FpPointerStructType = array[0..8] of Byte;
  FpPointerType = ^FpPointerStructType;
```

```
  PositionType  = record
    X: FloatType;
    Y: FloatType;
  end;
  SensorDataType = record
    Ref: PositionType;
    Adj: PositionType;
  end;

var
  FirstVar: Byte;   { must be first variable defined }
  ch,key: Char;
  temp,cmd,TextCommand,DefaultLine: String;
  OperationMnemonics,Operand: String;
  StereoVersion: str16;
  count,i,j,remline,Er,OpNum,code,code2: Integer;
  SensorNum,IOerr,DefIndex: Integer;
  GeoFile,DefaultFile: text;
  tempX,tempY,X1,Y1,X2,Y2,oldX2,oldY2: FloatType;
  arg,beginXpos,beginYpos,Xpos,Ypos,oldX,oldY:
FloatType;
  FirstX,NegX,NegY,SwapXY,SwapBegin: Boolean;
  first,VelocityFlag,DebugMode,Redrawing,done: Boolean;
  OkayForRedrawCommand,MultiTimeBaseFlag: Boolean;
 _Step_Size,_Step_Period,_Scan_Delay,_Jump_Period,
_Jump_Delay: Word;
  _Jump_Size,_Rivet_Reduction: FloatType;
  _Laser_On_Delay,_Laser_Off_Delay: Word;
  _Redraw_Count,_Redraw_Delay,_Redraw_Size: Word;
  KeyAbort,VersionCheckFlag,GeoCorrectFlag,DoingRivet:
Boolean;
  DriftCorrectionFlag,BinaryFlag: Boolean;
  changeX,changeY,ChangeV,BrokenX,BrokenY: FloatType;
  deltaX,deltaY,steps,GeoDeltaX,GeoDeltaY: FloatType;
  Distance,IntDeltaX,IntDeltaY: Integer;
  LongDeltaX,LongDeltaY,LongSteps,TotalSteps,TempSteps:
LongInt;
```

```
    RivetMainPeriod,CurrentRedrawCount,_Rivet_Count: Word;
    StoreCount,QueueCount: Word;
    angX,angY,tanX,GeoScale,GeoFact: FloatType;
    FieldX,FieldY,StartX,StartY,EndX,EndY: FloatType;
    GeoEndX,GeoEndY,Xcoord,Ycoord: FloatType;
    DiscreteX,DiscreteY: Integer;
    LongDiscreteX,LongDiscreteY: LongInt;
    DiscreteSteps: Word;
    regs: Registers;
    XC,YC,XCount,YCount,CoordCount,TableDivisor: Word;
    StereoSize: LongInt;
    CommandOfs,CommandSeg: Word;
    data,Return: Byte;
    XGeoCorrectTable,YGeoCorrectTable: GeoCorrectTablePtr;
    Sensor: array[1..2] of SensorDataType;
    DriftGain: PositionType;
    LastVar: Byte;   { must be last variable defined }

const
    {$IFDEF COP}
    ThisStereoVersion = '3.20 C ';
    {$ELSE}
    ThisStereoVersion = '3.20 NC';
    {$ENDIF}
    MinSystemVersion  = '3.20';
    CtrlBreak = #0;
    HexDigits: str16 = '0123456789ABCDEF';
    sp40 = '                                        ';
    Indent1 = 4;
    Indent2 = 41;
    AllBits = $FFFF;
    BitValue: BitArray =
      ($0001, $0002, $0004, $0008, $0010, $0020, $0040,
$0080, $0100, $0200, $0400, $0800, $1000, $2000, $4000,
$8000);
    Shutter = 0;  { control word bit 0 - shutter
open/closed }
```

```
  _BIOSVECTOR    = $16;
  _STEREOVECTOR = $65;
  MaxGeo     = 100;     { maximum GeoCorrectTable Index }
  GeoMult    = 4096;    { GeoCorrectTable LongInt
multiplier }
  StepRound = 0.02;     { rounding value for # of laser
steps }

label ExitProgram;

function StartOfProgramCode: String;
begin
  StartOfProgramCode :=
    'STEREO Device Driver Version 3.20 Copyright (C)
1988 by Tarnz Technologies ';
end;

var
  StepPeriodsArray,RivetPeriodsArray: _StepPeriodsType;

function __StereoOk: Boolean;
type
  CopyType  = array[1..20] of char;
var
  CopyPtr: ^CopyType;
  CopyStr: string[20];
begin
  with regs do begin
    AX := ($35 shl 8) or _STEREOVECTOR;
    MsDos(regs);
    CopyPtr := Ptr(ES,57);
  end;
  move(CopyPtr^[1],CopyStr[1],18);
  CopyStr[0] := #18;
  if CopyStr = 'Tarnz Technologies' then begin
    __StereoOk := true;
    with regs do
```

```
      CopyPtr := Ptr(ES,30);
    move(CopyPtr^[1],StereoVersion[1],4);
    StereoVersion[0] := #4;
  end else begin
    __StereoOk := false;
    StereoVersion := '?.??';
  end;
end;

function upper(text: String): String;
var
  i: Integer;
  temp: String;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): String;
var
  temp: String;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
```

```
end;

function CvtDelta(num: FloatType): Integer;
begin
  num := num * 256;
  if num < 32768.0 then CvtDelta := LibTrunc(num)
    else if num > 32768.0 then CvtDelta := LibTrunc(num)
      else CvtDelta := $7FFF;
end;

function CvtLongDelta(num: FloatType): LongInt;
var
  tnum: LongInt;
begin
  CvtLongDelta := LibTrunc(num) shl 16 +
LibTrunc(frac(num) * 65536);
end;

function sgn(num: FloatType): Integer;
begin
  if num<0 then sgn := -1
    else if num>0 then sgn := 1
      else sgn := 0;
end;

function CvtTicks(time: FloatType): Word;
begin
  CvtTicks := LibTrunc(time * 11.9);   {119.0 / 10.0 }
end;

procedure SkipToNum(var str: String);
var
  i: Integer;
begin
  i := 1;
  while (i <= length(str)) and not(str[i] in
['0'..'9','.']) do inc(i);
```

```
  if i>1 then str := copy(str,i,99);
end;

procedure RealParse(var RealNum: FloatType);
var
  i: Integer;
begin
  i := 1;
  while (i <= length(temp)) and (temp[i] = ' ') do
inc(i);
  delete(temp,1,i-1);
  if copy(temp,1,1) = '.' then insert('0',temp,1);
  val(temp,RealNum,code);
  if code<>0 then
val(copy(temp,1,code-1),RealNum,code2);
  temp := copy(temp,code+1,99);
end;

procedure IntParse(var IntNum: Word);
var
  num: FloatType;
begin
  if length(temp)=0 then IntNum := 0
  else begin
    val(temp,num,code);
    if code<>0 then val(copy(temp,1,code-1),num,code2);
    IntNum := LibTrunc(num);
    if code=0 then temp := ''
      else temp := copy(temp,code+1,99);
  end;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
```

```
  IntStr := temp;
end;

function RealStr(num: FloatType): str16;
var
  temp: str16;
begin
  if num<0 then Str(65536.0+num:6:0,temp)
    else Str(num:6:0,temp);
  while temp[1]=' ' do temp := copy(temp,2,6);
  RealStr := temp;
end;

function spc(len: Integer): str80;
begin
  spc := copy(sp40+sp40+sp40,1,len);
end;

procedure msg(text: String);
begin
(* return text to calling program *)
end;

procedure msgln(text: String);
begin
(* return text + Cr to calling program *)
end;

function MemAbs(pt: Pointer): LongInt;
begin
  MemAbs := LongInt(seg(pt^)) shl 4 + ofs(pt^);
end;

{
  converts a Single type (4 bytes) number to a Real (6
bytes)
}
```

```
procedure ConvertStoR(s,r: FpPointerType);
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;

function CharArgument: Char;
begin
  if BinaryFlag then begin
    CharArgument := char(Mem[CommandSeg:CommandOfs]);
    inc(CommandOfs);
  end else begin
    if length(Operand) > 0 then CharArgument :=
Operand[1]
      else CharArgument := #0;
    delete(Operand,1,1);  { slow, avoid for long
strings }
  end;
end;

function WordArgument: Word;
var
  WordArg: Word;
  code,code2: Integer;
begin
  if BinaryFlag then begin
    WordArgument := Mem[CommandSeg:CommandOfs] shl 8 +
                    Mem[CommandSeg:CommandOfs + 1];
    inc(CommandOfs,2);
  end else begin
    val(Operand,WordArg,code);
    if code<>0 then val(Operand,WordArg,code2);
```

```
begin
  Return := Er;
end;

function TryCommand: Boolean;
begin
  if Er=3 (queue full) then TryCommand := true
    else TryCommand := false;
  if keypressed (abort) then begin
    if StatusKey = CtrlBreak then begin
      TryCommand := false;
      KeyAbort := true;
    end;
  end;
end;

function TryBlockCommand: Boolean;
begin
  if (Er=3) (command queue full) or (Er=6) (exec queue
full)
    then TryBlockCommand := true
      else TryBlockCommand := false;
  if keypressed (abort) then begin
    if StatusKey = CtrlBreak then begin
      TryBlockCommand := false;
      KeyAbort := true;
    end;
  end;
end;

procedure newX(num: FloatType);
begin
  oldX := Xpos;
  Xpos := num;
end;

procedure newY(num: FloatType);
```

```
begin
  oldY := Ypos;
  Ypos := num;
end;

{
    AxesControl -- adjust axes
}
procedure AxesControl(oldX,oldY: FloatType;var
retX,retY: FloatType);
begin
  if SwapXY then begin
    retY := oldX;
    retX := oldY;
  end else begin
    retX := oldX;
    retY := oldY;
  end;
  if NegX then retX := 65535.0 - retX;
  if NegY then retY := 65535.0 - retY;
end;


procedure SetRedrawCount(RedrawCount: Integer);
begin
  if RedrawCount<>CurrentRedrawCount then begin
   CurrentRedrawCount := RedrawCount;
    Er := 3;
    while TryCommand do
      Er := __SetRedrawCount(CurrentRedrawCount);
    inc(QueueCount,3);
    SIHerror;
  end;
end;

{
   Laser Motion Routines:
```

```
    * ResetRedraw   - just mark for beginning of block
    * PerformRedraw - redraw and mark last section
    * MoveLaser     - new move alg using redraw & vector
slicing
    * JumpLaser     - controlled breakup of jump motion
to limit overshoot
}

procedure ResetRedraw;
begin
  if OkayForRedrawCommand and __BlockModeOn then begin
    if DebugMode then msgln('Mark
'+IntStr(_Redraw_Size));
    Er := 3;
    while TryCommand do
      Er := __Mark;
    inc(QueueCount);
    SIHerror;
  end;
  Distance := 0;
  Redrawing := false;
  OkayForRedrawCommand := false;
end;

procedure PerformRedraw;
begin
  if OkayForRedrawCommand and __BlockModeOn then begin
    if DebugMode then msgln('Redraw '+IntStr
(_Redraw_Size));
    Er := 3;
    while TryCommand do
      Er := __Redraw;
    inc(QueueCount);
    SIHerror;
  end;
  Distance := 0;
```

```
  Redrawing := false;
  OkayForRedrawCommand := false;
end;

{
    CorrectForDrift -- develop drift gain terms
}
procedure CorrectForDrift;
var
  quotient: FloatType;
begin
  quotient := Sensor[2].Ref.X - Sensor[1].Ref.X;
  if quotient <> 0 then
    DriftGain.X := (Sensor[2].Adj.X - Sensor[1].Adj.X) /
quotient
      else DriftGain.X := 0;
(*
  writeln('CorrectForDrift-X:');
  writeln('2Xadj=',Sensor[2].Adj.X:8:2,' -
1Xadj=',Sensor[1].Adj.X:8:2,' /');
  writeln('2Xref=',Sensor[2].Ref.X:8:2,' -
1Xref=',Sensor[1].Ref.X:8:2);
  writeln('DriftGainX=',DriftGain.X:8:2);
*)
  quotient := Sensor[2].Ref.Y - Sensor[1].Ref.Y;
  if quotient <> 0 then
    DriftGain.Y := (Sensor[2].Adj.Y - Sensor[1].Adj.Y) /
quotient
      else DriftGain.Y := 0;
(*
  writeln('CorrectForDrift-Y:');
  writeln('2Yadj=',Sensor[2].Adj.Y:8:2,' -
1Yadj=',Sensor[1].Adj.Y:8:2,' /');
  writeln('2Yref=',Sensor[2].Ref.Y:8:2,' -
1Yref=',Sensor[1].Ref.Y:8:2);
  writeln('DriftGainY=',DriftGain.Y:8:2);
*)
```

```
end;

{
    GeoCorrect -- geometric correct coordinate pair
}
procedure GeoCorrect(oldX,oldY: FloatType;var newX,newY:
FloatType);
var
  GX11,GX12,GX21,GX22,GY11,GY12,GY21,GY22: FloatType;
  Xf,Yf,XfYf: FloatType;
  XIndex,YIndex,Lookup: Word;
begin
  if GeoCorrectFlag then begin
    XIndex := LibTrunc(oldX / TableDivisor);
    if XIndex >= XCount then XIndex := XCount-1;
    YIndex := LibTrunc(oldY / TableDivisor);
    if YIndex >= YCount then YIndex := YCount-1;
    Lookup := XIndex * YCount + YIndex;
    GX11 := XGeoCorrectTable^[Lookup];
    GX12 := XGeoCorrectTable^[Lookup+1];
    GX21 := XGeoCorrectTable^[Lookup+YCount];
    GX22 := XGeoCorrectTable^[Lookup+YCount+1];
    GY11 := YGeoCorrectTable^[Lookup];
    GY12 := YGeoCorrectTable^[Lookup+1];
    GY21 := YGeoCorrectTable^[Lookup+YCount];
    GY22 := YGeoCorrectTable^[Lookup+YCount+1];
    Xf := (oldX - TableDivisor * 1.0 * XIndex) /
TableDivisor;
    Yf := (oldY - TableDivisor * 1.0 * YIndex) /
TableDivisor;
    XfYf := Xf * Yf;
    newX := (GX11 + Xf * (GX21 - GX11) + Yf * (GX12 -
GX11)
      + XfYf * (GX22 + GX11 - GX12 - GX21)) / GeoMult;
    newY := (GY11 + Xf * (GY21 - GY11) + Yf * (GY12 -
GY11)
      + XfYf * (GY22 + GY11 - GY12 - GY21)) / GeoMult;
```

```
  end else begin
    newX := oldX;
    newY := oldY;
  end;
  if DriftCorrectionFlag then begin
    newX := (newX - Sensor[1].Ref.X) * DriftGain.X +
Sensor[1].Adj.X;
    newY := (newY - Sensor[1].Ref.Y) * DriftGain.Y +
Sensor[1].Adj.Y;
  end;
end;

procedure MoveLaser;
begin
  AxesControl(Xpos,Ypos,EndX,EndY);
  AxesControl(oldX,oldY,StartX,StartY);
  changeX := EndX - StartX;
  changeY := EndY - StartY;
  if VelocityFlag then changeV := sqrt(sqr(changeX) +
sqr(changeY))
    else if abs(changeX) < abs(changeY) then ChangeV :=
abs(changeY)
        else ChangeV := abs(changeX);
  steps := ChangeV / _Step_Size;
  if steps>=1 then begin
    deltaX := changeX / steps;
    deltaY := changeY / steps;
    LongSteps  := LibTrunc(steps+StepRound);
    IntDeltaX := CvtDelta(deltaX);
    IntDeltaY := CvtDelta(deltaY);
    TotalSteps := 0;
    repeat
      BrokenX := StartX + deltaX * TotalSteps;
      BrokenY := StartY + deltaY * TotalSteps;
      TempSteps := LongSteps - TotalSteps;
      if TempSteps>256 then TempSteps := 256;
```

```
      if (CurrentRedrawCount > 1) and (_Redraw_Size>0)
then
        if Distance+TempSteps >= _Redraw_Size then begin
          TempSteps := _Redraw_Size - Distance;
          Redrawing := true;
        end;
      TotalSteps := TotalSteps + TempSteps;
      Distance := Distance + TempSteps;
      GeoCorrect(BrokenX,BrokenY,FieldX,FieldY);

{ Geometric Correction addition }
      if GeoCorrectFlag then begin
        GeoCorrect(BrokenX+TempSteps*DeltaX,BrokenY+
TempSteps*DeltaY,
          GeoEndX,GeoEndY);
        changeX := GeoEndX - FieldX;
        changeY := GeoEndY - FieldY;
        ChangeV := sqrt(sqr(changeX) + sqr(changeY));
        steps := ChangeV / _Step_Size;
        GeoDeltaX := ChangeX / steps;
        GeoDeltaY := ChangeY / steps;
        DiscreteX := CvtDelta(GeoDeltaX);
        DiscreteY := CvtDelta(GeoDeltaY);
        DiscreteSteps := LibTrunc(steps+StepRound);
      end else begin
        DiscreteX := IntDeltaX;
        DiscreteY := IntDeltaY;
        DiscreteSteps := TempSteps;
      end;

      Er := 3;  { send out move command }
      while TryCommand do begin
        Er :=
__AddVector(LibTrunc(FieldX),LibTrunc(FieldY),
                DiscreteX,DiscreteY,DiscreteSteps);
        OkayForRedrawCommand := true;
      end;
```

```
      inc(QueueCount,13);
      SIHerror;
      if Redrawing then PerformRedraw;
    until TotalSteps = LongSteps;
  end;
end;

procedure SingleTimebaseJumpLaser;
begin
  if _Jump_Delay<>0 then begin
    AxesControl(Xpos,Ypos,EndX,EndY);
    AxesControl(oldX,oldY,StartX,StartY);
    changeX := EndX-StartX;
    changeY := EndY-StartY;
    if VelocityFlag then changeV :=
sqrt(sqr(changeX)+sqr(changeY))
      else if abs(changeX) < abs(changeY) then ChangeV
:= abs(changeY)
          else ChangeV := abs(changeX);
    steps := ChangeV / _Jump_Size;
    if steps>0 then begin
      deltaX := changeX / steps;
      deltaY := changeY / steps;
      LongSteps  := LibTrunc(steps+StepRound) + 1;
      TotalSteps := 1;
      repeat
        BrokenX := StartX + deltaX*TotalSteps;
        BrokenY := StartY + deltaY*TotalSteps;
        TotalSteps := TotalSteps + 1;
        if TotalSteps >= LongSteps then begin
          BrokenX := EndX;
          BrokenY := EndY;
        end;
        GeoCorrect(BrokenX,BrokenY,FieldX,FieldY);
        Er := 3;                    { add discrete jump }
        while TryCommand do begin
```

```
          Er := __AddVector(LibTrunc(FieldX),
LibTrunc(FieldY),
                  0,0,1);        { incl a 1 tick pause }
          OkayForRedrawCommand := true;
        end;
        inc(QueueCount,13);
        SIHerror;
      until TotalSteps >= LongSteps;
    end;
    Er := 3;
    while TryCommand do
      Er := __AddPause(_Jump_Delay);
    inc(QueueCount,3);
    SIHerror;
  end;
end;

procedure MultiTimebaseJumpLaser;
begin
  AxesControl(Xpos,Ypos,EndX,EndY);
  AxesControl(oldX,oldY,StartX,StartY);
  changeX := EndX - StartX;
  changeY := EndY - StartY;
  if VelocityFlag then changeV := sqrt(sqr(changeX) +
sqr(changeY))
    else if abs(changeX) < abs(changeY) then ChangeV :=
abs(changeY)
        else ChangeV := abs(changeX);
  steps := ChangeV / _Jump_Size;
  if steps>=1 then begin
    if MultiTimebaseFlag then begin
      Er := 3;
      while TryCommand do
        Er := __SelectTimerChannel(2);
      inc(QueueCount,2);
      SIHerror;
    end;
```

```
    deltaX := changeX / steps;
    deltaY := changeY / steps;
    LongSteps  := LibTrunc(steps+StepRound);
    LongDeltaX := CvtLongDelta(deltaX);
    LongDeltaY := CvtLongDelta(deltaY);
    GeoCorrect(StartX,StartY,FieldX,FieldY);
    GeoCorrect(EndX,EndY,GeoEndX,GeoEndY);

{ Geometric Correction addition }
    if GeoCorrectFlag then begin
      changeX := GeoEndX - FieldX;
      changeY := GeoEndY - FieldY;
      ChangeV := sqrt(sqr(changeX) + sqr(changeY));
      steps := ChangeV / _Jump_Size;
      GeoDeltaX := ChangeX / steps;
      GeoDeltaY := ChangeY / steps;
      LongDiscreteX := CvtLongDelta(GeoDeltaX);
      LongDiscreteY := CvtLongDelta(GeoDeltaY);
      DiscreteSteps := LibTrunc(steps+StepRound);
    end else begin
      LongDiscreteX := LongDeltaX;
      LongDiscreteY := LongDeltaY;
      DiscreteSteps := LongSteps;
    end;
    Er := 3;  { send out jump as fast move command }
    while TryCommand do
      Er := __AddLongVector(LibTrunc(FieldX),
LibTrunc(FieldY),
              LongDiscreteX,LongDiscreteY,
DiscreteSteps);
    inc(QueueCount,19);
    SIHerror;
    if _Jump_Delay > 0 then begin
      Er := 3;  { send out jump as fast move command }
      while TryCommand do
        Er := __AddVector(LibTrunc(GeoEndX),LibTrunc
(GeoEndY),0,0,1);
```

```
        inc(QueueCount,13);
        SIHerror;
        if _Jump_Delay > 1 then begin
          Er := 3;
          while TryCommand do
            Er := __AddPause(_Jump_Delay-1);
          inc(QueueCount,3);
          SIHerror;
        end;
      end;
      if MultiTimebaseFlag then begin
        Er := 3;
        while TryCommand do
          Er := __SelectTimerChannel(1);
        inc(QueueCount,2);
        SIHerror;
      end;
    end;
  end;

  procedure JumpLaser;
  begin
    if MultiTimebaseFlag then MultiTimebaseJumpLaser
      else SingleTimebaseJumpLaser;
  end;

  {
     -  -  -  -  -
  }

  procedure BeginX;
  begin
    beginXpos := PositionArgument;
  end;

  procedure BeginY;
  var
```

```
  X1,Y1,X2,Y2: FloatType;
begin
  beginYpos := PositionArgument;
  if SwapBegin then AxesControl(beginXpos,beginYpos,
X1,Y1)
    else begin
      X1 := beginXpos;
      Y1 := beginYpos;
    end;
  GeoCorrect(X1,Y1,X2,Y2);
  Er := __SetHomePosition(LibTrunc(X2),LibTrunc(Y2));
  SIHerror;
end;

procedure BeginPosition;
begin
  BeginX;
  BeginY;
end;

procedure JumpX;
begin
  newX(PositionArgument);
end;

procedure JumpY;
begin
  newY(PositionArgument);
  JumpLaser;
end;

procedure PositionJump(X,Y: FloatType);
begin
  NewX(X);
  NewY(Y);
  if _Jump_Delay<>0 then begin
    JumpLaser;
```

```
    Er := 3;
    while TryCommand do
      Er := __AddPause(_Jump_Delay);
    inc(QueueCount,3);
    SIHerror;
  end;
end;

procedure JumpPosition;
begin
  JumpX;
  JumpY;
end;

procedure NextX;
begin
  newX(PositionArgument);
end;

procedure NextY;
begin
  newY(PositionArgument);
  MoveLaser;
end;

procedure PositionMove(X,Y: FloatType);
begin
  newX(X);
  newY(Y);
  MoveLaser;
end;

procedure MoveToPosition;
begin
  NextX;
  NextY;
end;
```

```
procedure StepSize;
begin
  _Step_Size := WordArgument;
  if _Step_Size < 1 then _Step_Size := 1;    { limit }
end;

procedure StepPeriods;
var
  i: Integer;
begin
  PerformRedraw;
  if BinaryFlag then begin
    count := WordArgument;
    for i:=0 to count-1 do StepPeriodsArray.periods[i]
:= TimeArgument;
  end else begin
    count := 0;
    while (Operand <> '') and (count < 10) do begin
      StepPeriodsArray.periods[count] := TimeArgument;
      inc(count);
    end;
  end;
  StepPeriodsArray.num := count;
  if count>0 then begin
(*    if DebugMode then begin
      msg('New step periods array: ');
      for i:=0 to count-1 do msg
(StepPeriodsArray.periods[i],' ');
      msgln('');
    end; *)
    Er := 3;
    if __BlockModeOn then begin
      while TryCommand do
        Er := __SetStepPeriods(StepPeriodsArray);
      inc(QueueCount,count*2+2);
    end else begin
```

```
      while TryCommand do
        Er := __AddChangeTimebase
(StepPeriodsArray.periods[0]);
      inc(QueueCount,3);
    end;
    SIHerror;
  end;
  PerformRedraw;
end;

procedure ScanDelay;
begin
  _Scan_Delay := WordArgument;
end;

procedure JumpPeriod;
begin
  if MultiTimebaseFlag then begin
    _Jump_Period := TimeArgument;
    Er := 3;
    while TryCommand do
      Er := __SelectTimerChannel(2);
    inc(QueueCount,2);
    SIHerror;
    Er := 3;
    while TryCommand do
      Er := __AddChangeTimebase(_Jump_Period);
    inc(QueueCount,3);
    SIHerror;
    Er := 3;
    while TryCommand do
      Er := __SelectTimerChannel(1);
    inc(QueueCount,2);
    SIHerror;
  end;
end;
```

```
procedure JumpSize;
begin
  _Jump_Size := WordArgument;
  if _Jump_Size < 1 then _Jump_Size := 1;    { limit }
end;

procedure JumpDelay;
begin
  _Jump_Delay := WordArgument;
end;

procedure LaserOn;
begin
  _Laser_On_Delay := TimeArgument;
  Er := __SetTurnOnDelay(_Laser_On_Delay);
  SIHerror;
end;

procedure LaserOff;
begin
  _Laser_Off_Delay := TimeArgument;
  Er := __SetTurnOffDelay(_Laser_Off_Delay);
  SIHerror;
end;

procedure SequenceMode;
begin
  { no-operation }
end;

procedure AbsoluteMode;
begin
  { no-operation }
end;

procedure Execute;
begin
```

```
  PerformRedraw;
  Er := 3;
  while TryBlockCommand do
    Er := __Execute;
  SIHerror;
  ResetRedraw;
end;

procedure ExecuteClear;
begin
  PerformRedraw;
  Er := 3;
  while TryBlockCommand do
    Er := __ExecuteClear;
  QueueCount := 0;
  SIHerror;
  ResetRedraw;
end;

procedure ClearTable;
begin
  Er := __Clear;
  QueueCount := 0;
  if Er<>5 then SIHerror;  { not in block mode okay }
  ResetRedraw;
end;

procedure SetMode;
begin
  if BinaryFlag then begin
    case CharArgument of
      'A': Er := __SetAutoShutterMode(CharArgument);
      'B': Er := __SetBlockMode(CharArgument);
      'P': Er := __SetPosAckMode(CharArgument);
    end;
  end else begin
    if length(TextCommand)=5 then case TextCommand[3] of
```

```
        'A': Er := __SetAutoShutterMode(TextCommand[5]);
        'B': Er := __SetBlockMode(TextCommand[5]);
        'P': Er := __SetPosAckMode(TextCommand[5]);
      end;
    end;
    SIHerror;
  end;

  procedure SoftReset;
  begin
    Er := __SetAutoShutterMode('-');   { off }
    SIHerror;
    Er := __SetBlockMode('-');         { off }
    SIHerror;
    Er := __SetPosAckMode('-');        { off }
    SIHerror;
    QueueCount := 0;
  end;

  procedure HardReset;
  begin
    SoftReset;
    Er := __ResetCmdQueue;
    SIHerror;
  end;

  procedure RedrawCount;
  begin
    _Redraw_Count := WordArgument;
    if _Redraw_Count < 1 then _Redraw_Count := 1
      else if _Redraw_Count > 10 then _Redraw_Count := 10;
    SetRedrawCount(_Redraw_Count);
    PerformRedraw;
    Er := 3;
    while TryCommand do
      Er := __SetRedrawCount(_Redraw_Count);
    inc(QueueCount,3);
```

```
  SIHerror;
end;

procedure RedrawDelay;
begin
  _Redraw_Delay := WordArgument;
  Er := 3;
  while TryCommand do
    Er := __SetRedrawDelay(_Redraw_Delay);
  inc(QueueCount,3);
  SIHerror;
end;

procedure RedrawSize;
begin
  _Redraw_Size := WordArgument;
end;

procedure RivetCount;
begin
  _Rivet_Count := WordArgument;
  if _Rivet_Count < 1 then _Rivet_Count := 1
    else if _Rivet_Count > 10 then _Rivet_Count := 10;
  _Rivet_Count := _Rivet_Count - 1;
end;

procedure RivetPeriods;
var
  i: Integer;
begin
  if BinaryFlag then begin
    count := WordArgument;
    RivetMainPeriod := TimeArgument;
    for i:=0 to count-1 do RivetPeriodsArray.periods[i]
:= TimeArgument;
  end else begin
    RivetMainPeriod := TimeArgument;
```

```
    count := 0;
    while (Operand <> '') and (count<10) do begin
      RivetPperiodsArray.periods[count] := TimeArgument;
      inc(count);
    end;
  end;
  RivetPeriodsArray.num := count;
(*  if DebugMode then begin
    msg('New rivet periods array: ');
    msg(' Main=',RivetMainPeriod);
    for i:=0 to count-1 do msg('
',RivetPeriodsArray.periods[i]);
    msgln('');
  end; *)
end;

procedure RivetReduction;
begin
  _Rivet_Reduction := WordArgument;
end;

procedure ResetFromRivets;
begin
  if RivetPeriodsArray.num>0 then begin
    { set standard step periods }
    Er := 3;
    while TryCommand do
      Er := __SetStepPeriods(StepPeriodsArray);
    inc(QueueCount,StepPeriodsArray.num*2+2);
    SIHerror;
  end;
end;

procedure RivetX;
begin
  newX(PositionArgument);
end;
```

```
procedure RivetY;
var
  StartX,StartY,EndX,EndY: FloatType;
  ReducedStartX,ReducedStartY,ReducedEndX,ReducedEndY:
FloatType;
  ReductionX,ReductionY: FloatType;
  DirX,DirY,ReducedDirX,ReducedDirY: Integer;
begin
  StartX := oldX;
  StartY := Ypos;
  EndX   := Xpos;
  EndY   := PositionArgument;
  { set rivet step periods }
  Er := 3;
  while TryCommand do
    Er := __AddChangeTimeBase(RivetMainPeriod);
  inc(QueueCount,3);
  SIHerror;
  { draw complete vector }
  NextY;
  DirX := sgn(EndX - StartX);
  DirY := sgn(EndY - StartY);
  ReductionX := _Rivet_Reduction * deltaX;
  ReductionY := _Rivet_Reduction * deltaY;
  ReducedStartX := StartX + ReductionX;
  ReducedStartY := StartY + ReductionY;
  ReducedEndX   := EndX   - ReductionX;
  ReducedEndY   := EndY   - ReductionY;
  ReducedDirX := sgn(ReducedEndX - ReducedStartX);
  ReducedDirY := sgn(ReducedEndY - ReducedStartY);
  DoingRivet := true;
  { do rivet if reduced vector size is greater than 0
 units }
  if ((ReducedDirX = DirX) and (ReducedDirY = DirY))
    and (_Rivet_Count > 0) then begin
    { end standard redraw if any }
```

```
    PerformRedraw;
    if RivetPeriodsArray.num>0 then begin
      { set rivet step periods }
      Er := 3;
      while TryCommand do
        Er := __SetStepPeriods(RivetPeriodsArray);
      inc(QueueCount,RivetPeriodsArray.num*2+2);
      SIHerror;
    end;
    { set rivet pass count }
    SetRedrawCount(_Rivet_Count);
    { draw reduced vector - the rivet }
    PositionJump(ReducedStartX,ReducedStartY);
    PositionMove(ReducedEndX,ReducedEndY);
    { end rivet redraw }
    PerformRedraw;
    { restore original redraw pass count }
    SetRedrawCount(_Redraw_Count);
    { jump to end of full vector }
    PositionJump(EndX,EndY);
  end;
  { reset standard step periods }
  Er := 3;
  while TryCommand do
    Er := __SetStepPeriods(StepPeriodsArray);
  inc(QueueCount,StepPeriodsArray.num*2+2);
  SIHerror;
  DoingRivet := false;
end;

procedure RivetPosition;
begin
  RivetX;
  RivetY;
end;

procedure WaitIdle;
```

```
var
  time: Integer;
begin
  time := WordArgument;
  if time < 100 then time := 100;  { set minimum delay
time }
  delay(time);  { delay long enough for background task
to see EX }
  repeat
  until __GetBkgndIdleStatus or keypressed;
  if keypressed then if StatusKey = CtrlBreak then
KeyAbort := true;
end;

{
   Drift Correction Commands
}

procedure SetDrift;
begin
  if BinaryFlag then DriftCorrectionFlag :=
(CharArgument = '+')
    else DriftCorrectionFlag := (TextCommand[3] = '+');
end;

procedure Reference1;
begin
  Sensor[1].Ref.X := FloatArgument;
  Sensor[1].Ref.Y := FloatArgument;
end;

procedure Reference2;
begin
  Sensor[2].Ref.X := FloatArgument;
  Sensor[2].Ref.Y := FloatArgument;
end;
```

```
procedure DriftAdj1;
begin
  Sensor[1].Adj.X := FloatArgument;
  Sensor[1].Adj.Y := FloatArgument;
  CorrectForDrift;
end;

procedure DriftAdj2;
begin
  Sensor[2].Adj.X := FloatArgument;
  Sensor[2].Adj.Y := FloatArgument;
  CorrectForDrift;
end;


{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    STEREO Command Parsers

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure CallOperation(OpNum: Integer);
begin
  case OpNum of
     1: BeginX;          { BX }
     2: BeginY;          { BY }
     3: JumpX;           { JX }
     4: JumpY;           { JY }
     5: NextX;           { NX }
     6: NextY;           { NY }
     7: StepSize;        { SS }
     8: StepPeriods;     { SP }
     9: ScanDelay;       { SD }
    10: JumpPeriod;      { JP }
    11: JumpSize;        { JS }
    12: JumpDelay;       { JD }
    13: LaserOn;         { LO }
```

```
      14: LaserOff;          { LF }
      15: SequenceMode;      { NS }
      16: AbsoluteMode;      { AB }
      17: Execute;           { EX }
      18: ExecuteClear;      { EC }
      19: ClearTable;        { CL }
      20: SetMode;           { MD }
      21: SoftReset;         { SR }
      22: HardReset;         { HR }
      23: RedrawCount;       { RC }
      24: RedrawDelay;       { RD }
      25: RedrawSize;        { RS }
      26: RivetCount;        { VC }
      27: RivetPeriods;      { VP }
      28: RivetReduction;    { VR }
      29: RivetX;            { VX }
      30: RivetY;            { VY }
      31: WaitIdle;          { WI }
      32: SetDrift;          { DC }
      33: Reference1;        { R1 }
      34: Reference2;        { R2 }
      35: DriftAdj1;         { D1 }
      36: DriftAdj2;         { D2 }
      37: BeginPosition;     { BG }
      38: JumpPosition;      { JM }
      39: MoveToPosition;    { MV }
      40: RivetPosition;     { RV }
      else Return := 128;    { unrecognized command }
    end;
  end;

  procedure BinaryParse;
  var
    OpNum: Integer;
  begin
    { break apart multiple commands }
    done := false;
```

```
  while (not KeyAbort) and (not done) and (Return = 0)
do begin
    OpNum := Mem[CommandSeg:CommandOfs];
    if OpNum <> 0 then begin    { cmd of 0 marks end }
      inc(CommandOfs);
      CallOperation(OpNum);
    end else done := true;
  end;
end;

procedure ParseTextCommand;
var
  OpName: str4;
begin
  OpName := copy(TextCommand,1,2);
  Operand := copy(TextCommand,3,99);
  i := pos(OpName,OperationMnemonics);
  CallOperation((i + 2) div 4);    { good cmds give
1,2,3... }
end;

procedure TextParse;
begin
  { break apart multiple command strings }
  TextCommand := '';
  done := false;
  while (not KeyAbort) and (not done) and (Return = 0)
do begin
    Ch := chr(Mem[CommandSeg:CommandOfs]);
    done := Ch < ' ';
    if done or (Ch=';') then begin
      inc(CommandOfs);
      ParseTextCommand;
      TextCommand := '';
    end else begin
      if Ch in ['!'..'~'] then TextCommand :=
TextCommand + upcase(Ch);
```

```
      inc(CommandOfs);
    end;
  end;
end;


{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    Process STEREO Commands coming through the Software
Interrupt

    - Software Interrupt $65 (101) Handler -

    ES:BX points to first character of string.  String
should end with a character value of 0, 10 (LF), or 13
(CR).  An error code will be returned in AL with 0
meaning no error occurred.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure
ReceiveCommand(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP:
Word);
interrupt;
begin
  KeyAbort := false;
  CommandOfs := BX;
  CommandSeg := ES;
  INLINE ($FB);   { STI    ;allow interrupts }

  Return := 0;
  QueueCount := 0;
  case AX of
    0: begin
         BinaryFlag := false;
         TextParse;
       end;
```

```
    1: begin
         BinaryFlag := true;
         BinaryParse;
       end;
    else Return := 129;    { invalid STEREO request }
  end;

  CX := QueueCount;
  AX := Return;
  BX := CommandOfs;
end;


{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    Memory Resident Routines

    After the following routines install the
memory-resident routines above, they are thrown away and
not used again.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

procedure ResidentTermination;
begin
  with regs do begin
    AH := $31;
    AL := 1;
    DX := LibTrunc(StereoSize) shr 4;
    MsDos(regs);
  end;
end;

procedure InitMemoryResidency;
begin
  SetIntVec(_STEREOVECTOR,@ReceiveCommand);
end;
```

```
{   -  -  -  -  -  -   }


function WhiteSpace(var str: String;loc: Integer):
Integer;
begin
  while (loc <= length(str)) and (str[loc] in [#9,' '])
do loc := loc + 1;
  WhiteSpace := loc;
end;

procedure CheckForOptions(var str: String);
var
  i: Integer;
begin
  str := upper(str);
  while (length(str) > 1) and (str[1] = ' ') do
delete(str,1,1);
  i := pos('/D',str);   { set debug mode }
  if i>0 then begin
    DebugMode := true;
    delete(str,i,2);
  end;
  i := pos('/NV',str);  { not constant vector velocity }
  if i>0 then begin     { larger of X and Y axes }
    VelocityFlag := false;
    delete(str,i,3);
  end;
  i := pos('/NG',str);  { no geometric correction }
  if i>0 then begin
    GeoCorrectFlag := false;
    delete(str,i,3);
  end;
  i := pos('/NMT',str);  { no multiple timebase }
  if i>0 then begin
```

```
    MultiTimebaseFlag := false;
    delete(str,i,4);
  end;
  i := pos('/NC',str);  { no version check }
  if i>0 then begin
    VersionCheckFlag := false;
    delete(str,i,3);
  end;
  i := pos('/SWAPXY',str);  { swap X & Y coordinates }
  if i>0 then begin
    SwapXY := true;
    delete(str,i,7);
  end;
  i := pos('/AXESBEGIN',str);  { axes-control begin
home) coordinates }
  if i>0 then begin
    SwapBegin := true;
    delete(str,i,10);
  end;
  i := pos('/GEO',str);  { geo-correct axes order }
  if i>0 then begin
    j := WhiteSpace(str,i+4);
    case str[j] of
      'X': FirstX := true;
      'Y': FirstX := false;
      'N': GeoCorrectFlag := false;
    end;
    delete(str,i,j+2-i);
  end;
  i := pos('/X',str);  { X-axis orientation }
  if i>0 then begin
    NegX := (str[i+2]='-');
    delete(str,i,3);
  end;
  i := pos('/Y',str);  { Y-axis orientation }
  if i>0 then begin
    NegY := (str[i+2]='-');
```

```
    delete(str,i,3);
  end;
end;

procedure ShowAxis(Neg: Boolean;AxCh: Char);
begin
  if Neg then write('-') else write('+');
  write(AxCh,' axis');
end;

procedure ShowGeometry;
begin
  write('Geometry is ');
  if SwapXY then write('XY Swapped, ');
  if SwapBegin then write('Begin Swapped, ');
  if FirstX then begin
    ShowAxis(NegX,'X');
    write(' first, then ');
    ShowAxis(NegY,'Y');
  end else begin
    ShowAxis(NegY,'Y');
    write(' first, then ');
    ShowAxis(NegX,'X');
  end;
  writeln('.');
  if GeoCorrectFlag then writeln('Geometric Correction
active.');
  if MultiTimebaseFlag then writeln('Multiple Timebase
active.');
end;

procedure ReadDefaultFile;
begin
  while not eof(DefaultFile) do begin
    readln(DefaultFile,DefaultLine);
    if (length(DefaultLine)>0) and (DefaultLine[1]<>'!')
then
```

```
      while length(DefaultLine)>0 do begin
        DefIndex := pos(';',DefaultLine);
        if DefIndex=0 then DefIndex :=
length(DefaultLine) + 1;
        TextCommand := copy(DefaultLine,1,DefIndex-1);
        ParseTextCommand;
        delete(DefaultLine,1,DefIndex);
      end; {while}
  end; {while not eof}
  close(DefaultFile);
end;


{  - - -  M A I N   P R O G R A M  - - -  }


begin
  CheckBreak := false;
  FillChar(FirstVar,ofs(LastVar)-ofs(FirstVar),#0);
{ clear variables }
  VelocityFlag        := true;   { default to constant
vector velocity }
  GeoCorrectFlag      := true;   { default to geometric
correction }
  DriftCorrectionFlag := true;   { perform drift
correction }
  VersionCheckFlag    := true;
  writeln;
  writeln(#213,many(#205,45),#184);
  writeln(#179,'  STEREO, Stereolithography Device
Driver    ',#179);
  writeln(#179,'  Version ',ThisStereoVersion,'
                ',#179);
  writeln(#179,'  Copyright (C) 1988 by Tarnz
Technologies   ',#179);
  writeln(#179,'  3D Systems Laser Stereolithography
System  ',#179);
```

```
  writeln(#212,many(#205,45),#190);
  writeln;
  cmd := StartOfProgramCode;
  ReadCommandLine(cmd);
  if not __LaserPresent then goto ExitProgram;
  if __StereoOk then begin
    writeln('STEREO Device Driver Version
',StereoVersion,
      ' already installed.');
    goto ExitProgram;
  end;
  CheckForOptions(cmd);
  if VersionCheckFlag then
    if LaserVersion < MinSystemVersion then begin
      writeln('Obsolete LASER system in use.');
      goto ExitProgram;
    end;
  MultiTimebaseFlag := __GetTimerCapability;  { from
LASER }
  Xpos       := 32767;
  Ypos       := 32767;
  DoingRivet := false;
  RivetMainPeriod       := 1190;
  StepPeriodsArray.num  := 0;
  RivetPeriodsArray.num := 0;
  OperationMnemonics := '*BX *BY *JX *JY *NX *NY *SS *SP
*SD *JP *JS *JD '
    + '*LO *LF *NS *AB *EX *EC *CL *MD *SR *HR *RC *RD
*RS *VC *VP *VR '
    + '*VX *VY *WI *DC *R1 *R2 *D1 *D2 *BG *JM *MV *RV';
  Sensor[1].Ref.X := 0;
  Sensor[1].Ref.Y := 0;
  Sensor[2].Ref.X := 65535;
  Sensor[2].Ref.Y := 65535;

  { read and/or set defaults }
  Assign(DefaultFile,'STEREO.DEF');
```

```
  {$I-}
  reset(DefaultFile);
  {$I+}
  if IOresult<>0 then begin
    writeln('Can''t find STEREO.DEF default file!');
    _Step_Size       := 2;
    _Step_Period     := 119;
    _Scan_Delay      := 64;
    _Jump_Period     := 50;
    _Jump_Size       := 1024;
    _Jump_Delay      := 0;
    _Laser_On_Delay  := 0;
    _Laser_Off_Delay := 0;
    _Redraw_Count    := 1;
    _Redraw_Delay    := 0;
    _Redraw_Size     := 200;
    _Rivet_Count     := 1;
    _Rivet_Reduction := 100;
  end else ReadDefaultFile;
  CurrentRedrawCount := _Redraw_Count;

  XGeoCorrectTable := HeapPtr;
  if GeoCorrectFlag then begin
   { read Geometric Correction Table }
    Assign(GeoFile,'STEREO.GEO');
    {$I-}
    reset(GeoFile);
    {$I+}
    if IOresult<>0 then begin
      writeln('Can''t find STEREO.GEO Geometric
Correction File!');
      GeoCorrectFlag := false;
    end else begin  { read geometric correction table
from file }
      repeat
        readln(GeoFile,temp);
        if upper(copy(temp,1,7)) = '!SENSOR' then begin
```

```
            val(temp[8],SensorNum,code);
            temp := copy(temp,9,99);
            if SensorNum in [1..2] then begin
              RealParse(Sensor[SensorNum].Ref.X);
              RealParse(Sensor[SensorNum].Ref.Y);
            end;
            temp := '!';
          end;
        until ((length(temp)>0) and (temp[1]<>'!')) or
eof(GeoFile);
        SkipToNum(temp);
        IntParse(XCount);
        SkipToNum(temp);
        IntParse(YCount);
        YGeoCorrectTable :=
PtrAdd(XGeoCorrectTable,LongInt(XCount*YCount) shl 2);
        SkipToNum(temp);
        IntParse(CoordCount);
        SkipToNum(temp);
        IntParse(TableDivisor);
        If TableDivisor < 1 then TableDivisor := 1024;
{ for V2.30 compatibility }
        if (XCount<>YCount) or (CoordCount<>2)
        or (XCount>MaxGeo) or (YCount>MaxGeo) then begin
          writeln('Illegal format for STEREO.GEO file.');
          halt(1);
        end;
        XC := 0;
        YC := 0;
        StoreCount := 0;
        while (XC<XCount) and not eof(GeoFile) do begin
          while (YC<YCount) and not eof(GeoFile) do begin
            repeat
              readln(GeoFile,temp);
            until ((length(temp)>0) and (temp[1]<>'!')) or
eof(GeoFile);
            SkipToNum(temp);
```

```
            RealParse(Xcoord);
            SkipToNum(temp);
            RealParse(Ycoord);
            XGeoCorrectTable^[StoreCount] :=
LibTrunc(Xcoord*GeoMult);
            YGeoCorrectTable^[StoreCount] :=
LibTrunc(Ycoord*GeoMult);
            inc(StoreCount);
            inc(YC);
          end; {while YC}
          inc(XC);
          YC := 0;
        end; {while XC}
        if XC < XCount then begin
          writeln('Not enough entries in STEREO.GEO
file.');
          halt(2);
        end;
        while not eof(GeoFile) do begin
          readln(GeoFile,temp);
          if (length(temp)>0) and (temp[1]<>'!') then
begin
            SkipToNum(temp);
            if temp[1] in ['0'..'9','.'] then begin
              writeln('Too many entries in STEREO.GEO
file.');
              halt(3);
            end;
          end;
        end;
        close(GeoFile);
      end;
    end;
    for i:=1 to 2 do Sensor[i].Adj := Sensor[i].Ref;
    CorrectForDrift;   { gain is 1, offset is 0 }
    StereoSize := MemAbs(XGeoCorrectTable) +
(LongInt(XCount*YCount) shl 3)
```

```
    - (LongInt(PrefixSeg) shl 4) + 4096;
  StereoSize := ((StereoSize shr 4)+1) shl 4;
(*
  writeln('MemAbs XGeo=',MemAbs(XGeoCorrectTable));
  writeln('MemAbs YGeo=',MemAbs(YGeoCorrectTable));
  writeln('X*Y shl 3=',LongInt(XCount*YCount) shl 3);
  writeln('PrefixSeg=',LongInt(PrefixSeg));
  writeln('StereoSize=',StereoSize);
*)
  writeln('STEREO Device Driver installed using ',
    RealStr(StereoSize/1024),'K bytes.');
  ShowGeometry;

  InitMemoryResidency;
  ResidentTermination;

ExitProgram:
end.
^Z
```

```
{=========================================================
   3D Systems Stereolithography System Software

=========================================================

   RECOAT1.PAS
   SLA-1 Control Program

---------------------------------------------------------
   Recent History:
   11/12/87  Ver 2.30  first Beta software release
---------------------------------------------------------}

{$M 16384,0,0}

uses Crt,Dos;

const
  VersionId = '3.20';
  LaserBoardAddress = $300;
  PowerPort     = $304;
  _INIT_OUT     = $01;  { outputs }
  _ZSTAGE_OUT   = $02;
  _SHUTTER_OUT  = $04;
  _LASER_OUT    = $08;
  _INIT_IN      = $10;  { inputs }
  _DOOR_IN      = $20;
  _SHUTTER_IN   = $40;
  _LASER_IN     = $80;
  Indent = 22;
  LASERVECTOR  = $64;
  STEREOVECTOR = $65;
  HomeDirectory = 'C:\3DSYS\';
  FixFileName   = 'FIX.DAT';
  ParamFileName = 'POWER.PRM';
  ErrorFileName = 'C:\SYSTEM.ERR';
  CtrlC = ^C;
```

```
  Zel: array[1..12] of Integer =
(1,4,3,6,1,4,6,2,5,0,3,5);
  DazeType: array[0..6] of String =
('Sunday','Monday','Tuesday','Wednesday','Thursday','Fri
day','Saturday');
  DayOfWeek : String[21] = 'SunMonTueWedThuFriSat';

type
  str4 = string[4];
  str16 = string[16];
  str10 = string[10];
  str80 = string[80];
  LaserPowerType = (_Unknown,_Level,_Toggle);
  OnOffType = (On,Off);

var
  key: Char;
  regs: Registers;
  ParamFile,ErrorFile,FixFile : text;
  LaserVersion,StereoVersion: str4;
  CurrentDate, CurrentTime : str80;
  temp,IndentStr,FixFileLine,ParamFileLine: String;
  StepIter,i,IOerr: Integer;
  sec,minu,hour,day,mon,year,weekday : Integer;
  QSize  : LongInt;
  LaserPower: LaserPowerType;
  LaserOff,ShowStuff: Boolean;
  SweepStep,PlungeStep : integer;
{
   Cursor on / off routines
}

function HexByte(num:Byte):str4;
const
   HexChars: str16= '0123456789ABCDEF';
begin
```

```
   HexByte := HexChars[num shr 4 + 1] + HexChars[num and
$0F +1];
end;

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;

{
  Show error and store it on disk
```

```
}
procedure SystemError(Text: string);
begin
  writeln('ERROR: '+Text);
  beep;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'POWER: ',Text);
  close(ErrorFile);
end;

{
  Describe Error to Operator
}
procedure PrintError(Text: String);
begin
  if whereX>1 then writeln;
  writeln('*** ',Text);
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
```

```
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = Ctrl C then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;
```

```
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                      {  Type          1    2    3    4   }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
```

```
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;

procedure GetSelection;
begin
  repeat
    HandleKeyboardRequest;
    key := upcase(key);
    if key in ['1'..'7','X'] then exit;
  until false;
end;

procedure ShowPowerMenu;
begin
  window(1,1,80,25);
  ClrScr;
  centerln('RECOAT1, SLA-1 Recoater Utility Control
Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  for i:=1 to 5 do writeln;
  DrawBox(Indent-1,8,80-Indent,17,1);
  window(Indent,9,80-Indent-1,16);
  ClrScr;
  writeln(' 1. Initialize Sweeper and Plunger ');
  writeln(' 2. Move Sweeper Up');
  writeln(' 3. Move Sweeper Down');
  writeln(' 4. Move Plunger Up');
  writeln(' 5. Move Plunger Down');
  writeln(' 6. VOID');
  writeln(' 7. VOID');
  write(' X. Exit');
  window(1,1,80,25);
  gotoxy(28,19);
```

```
  write('Please make a selection.');
  DrawBox(10,23,69,25,1);
end;

procedure UserWait;
begin
   writeln;
   writeln;
   writeln('       ... press any key to continue ');
   repeat delay(100) until KeyPressed;
   key:=Readkey;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;
```

```
function Zeller(month,day,year:integer):Integer;
begin
   if month<3 then dec(year);
   Zeller := ( Zel[month] + day + year + (year shr 2))
mod 7;
end;

procedure SetLaserOnOffTime;
  var
     ChronoFile  :  text;
     ChronoStr,ServiceStr   :  String;
     AbsDate     :  array[1..8] of char;
     AbsTime     :  array[1..8] of char;
     OffsetHour, OffsetMonth  :  array[1..2] of char;
     yearHold,nnnnstr    :  string[4];
     OffsetDay   :  array[1..3] of char;
     Fileexists, AM, fileempty, endoffile  : boolean;
     OnEntryExists, OffEntryExists : boolean;
     TryToAddOn, TryToAddOff       : boolean;
     j, nnn, valcode         :  integer;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetChipTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
```

```
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $02;   { get minutes }
          Minu := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $04;    { get hours }
            Hour := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $07;    { get day }
              Day := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $08;    { get month }
                Mon := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then begin
                  port[$70] := $09;    { get year }
                  Year := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then good :=
true;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
    Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
    Minu := (Minu shr 4) * 10 + (Minu and $0F);
    Hour := (Hour shr 4) * 10 + (Hour and $0F);
    Day  := (Day  shr 4) * 10 + (Day  and $0F);
    Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
    Year := (Year shr 4) * 10 + (Year and $0F);
```

```
  WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetChipTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str4;
  AmPm: Char;
begin
  GetChipTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
{  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear; }
  CurrentDate := DazeType[Zeller(Mon,Day,Year)]+'
'+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;
```

```
procedure ShowCurrentLDT;
begin
     ClrScr;
     write('      Current Laser Status = ');
     if port[PowerPort + 2] and _LASER_IN   = 0 then
writeln('On ')
     else writeln('Off');
     GetChipTimeStamp;
     writeln('      Current Date = ',CurrentDate);
     writeln('      Current Time = ',CurrentTime);
     writeln;
end;

function DayOfWeekZeller:integer;
var
   imonth, iday, iyear : integer;
   cmonth, cday, cyear : array[1..2] of char;
begin
   for i:= 1 to 2 do begin
      cmonth[i] := AbsDate[i];
      cday[i]   := AbsDate[i+3];
      cyear[i]  := AbsDate[i+6];
   end;
   val(cmonth,imonth,valcode);
   val(cday,iday,valcode);
   val(cyear,iyear,valcode);
   DayOfWeekZeller:= Zeller(imonth,iday,iyear);
end;


procedure IssueTimeCommand(laserCMD: str4);
   var
      CMDnoperiod : str4;
      iend        : integer;
begin
```

```
    if laserCMD = 'off.' then TryToAddOff:=true
                         else TryToAddOn :=true;
    CMDnoperiod:=laserCMD;
    CMDnoperiod[pos('.',laserCMD)]:=' ';
    ServiceStr:='/C SERVICE LASER ';
    ShowCurrentLDT;
    writeln;
    writeln('Enter:');
    writeln;
    writeln('      M  specify minutes from now for laser
',CMDnoperiod);
    writeln;
    writeln('      T  specify time and date for laser
',CMDnoperiod);
    repeat key:=upcase(Readkey) until key in ['T','M'];
    ShowCurrentLDT;
    if key='T' then begin
       writeln;
       writeln('Enter date and time to turn the laser
',laserCMD);
       writeln;
       writeln('FORMAT: mo/dd/yr hr:mn xx');
       writeln;
       writeln('  where mo=month  dd=day     yr=year');
       writeln('    hr=hour   mn=minute  xx=AM or PM ');
       writeln;
       CursorOn;
       Readln(chronoStr);
       iend:=0;
       iend:= pos('M',chronoStr);
       if iend=0 then iend:=pos('m',chronoStr);
       if iend=0 then begin
          ClrScr;
          gotoxy(1,3);
          writeln('   Invalid Entry: please specify AM
or PM. ');
          userwait;
```

```
            exit;
          end;
          if laserCMD='on. ' then ServiceStr := ServiceStr
+ 'ON DATE '
                              else ServiceStr := ServiceStr
+ 'OFF DATE ';
          for i:= 1 to pos(' ',chronoStr)
             do ServiceStr := ServiceStr+chronoStr[i];
          ServiceStr:= ServiceStr + 'TIME';
          for i:= pos(' ',chronoStr) to iend
             do ServiceStr := ServiceStr+chronoStr[i];
       end else begin
          writeln;
          writeln('Enter minutes from now when the laser
will turn ',laserCMD);
          writeln;
          writeln;
          writeln;
          CursorOn;
          Readln(chronoStr);
          val(chronoStr,nnn,valcode);
          if (valcode=0) and (nnn>0) and (nnn<999) then
begin
             Str(nnn:4,nnnnstr);
             for i:= 1 to 4 do if nnnnStr[i]=' ' then
nnnnStr[i]:='0';
             if laserCMD='on. ' then ServiceStr :=
ServiceStr + 'ON +'
                                 else ServiceStr :=
ServiceStr + 'OFF +';
             ServiceStr := ServiceStr + nnnnStr;
          end;
       end;
       CursorOff;
       ServiceStr:=ServiceStr + '/i/t/P';  { priority
service }
       exec('C:\COMMAND.COM',ServiceStr);
```

```
    delay(100);
end;  { procedure IssueTimeCommand }

begin
TryToAddOn := false;
TryToAddOff:= false;
repeat
  window(1,1,80,25);
  ClrScr;
  centerln('RECOAT1, SLA-1 Recoater Utility Control
Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');

  writeln;
  centerln('Laser On/Off Timer Menu');
  DrawBox(10,8,80-10,23,1);
  window(13,9,80-15,22);
  ClrScr;
  {$I-}
  assign(chronofile,'\Chrono.sys');
  reset(chronofile);
  fileexists := (IOResult = 0);
  if not fileexists then begin
     gotoxy(1,6);
     writeln('*** ERROR : CHRONO.SYS file not found.');
     UserWait;
     exit;
  end else begin
     ShowCurrentLDT;
     chronoStr:='';
     fileempty := true;
     endoffile := false;
     OnEntryExists := false;
     OffEntryExists := false;
     for j:=1 to 2 do begin
```

```
      if not endoffile then begin
      repeat
         Readln(chronofile,ChronoStr);
         endoffile:=EOF(chronofile);
      until endoffile or ((chronostr[1]='A') and
(chronostr[3]='L'));
      if (not endoffile or (chronostr[1]='A')) then
fileempty:=false;
      if (not fileempty) and (chronostr[1]='A') then
begin
         if chronoStr[4]='O' then begin
            write('The laser is set to turn on ');
            OnEntryExists:=true
         end else begin
            write('The laser is set to turn off ');
            OffEntryExists:=true
         end;
         for i:= 1 to 8 do begin
            AbsDate[i]:=chronoStr[i+5];
            AbsTime[i]:=chronoStr[i+14];
         end;
         for i:= 1 to 2 do begin
            OffsetDay[i]:=chronoStr[i+23];
            OffsetHour[i]:=chronoStr[i+27];
            OffsetMonth[i]:=chronoStr[i+30];
         end;
         OffsetDay[3]:=chronoStr[26];
         writeln('at ');
         if (OffsetDay<>'000') or (OffsetHour<>'00') or
(OffsetMonth<>'00')
         then begin
            write('      ');
            if OffsetMonth<>'00' then write(OffsetMonth,'
month(s) ');
            if OffsetHour <>'00' then write(OffsetHour,'
hour(s) ');
```

```
            if OffsetDay <>'000' then write(OffsetDay,'
day(s) ');
            writeln('from ');
          end;
          writeln('                ',AbsTime,' on
',DazeType[DayOfWeekZeller],', ',AbsDate,'. ');
          writeln;
        end; {repeat}
        end; {if not end of file}
        end; {for j:= 1 to 2 }
    end;
    close(chronofile);
    {$I+}
    if not TryToAddOn and not TryToAddOff
    and not OnEntryExists and not OffEntryExists then
begin
        writeln;
        writeln('      No entries found in the laser timer
queue.');
    end;
    if TryToAddOn and not OnEntryExists
    then writeln('Invalid ON entry ignored.');
    if TryToAddOff and not OffEntryExists
    then writeln('Invalid OFF entry ignored.');
    gotoxy(1,12);
    writeln('Select:  1. Remove ON entry      3. Add ON
entry ');
    writeln('         2. Remove OFF entry     4. Add OFF
entry');
    write  ('            ( X to exit )
    ');
    repeat key:=upcase(readkey) until key in
['1'..'4','X'];
    if key='X' then exit;
    ClrScr;
    TryToAddOn:=false;
    TryToAddOff:=false;
```

```
  if key='1' then exec('C:\COMMAND.COM','/C SERVICE
LASER ON CANCEL/i/t');
  if key='2' then exec('C:\COMMAND.COM','/C SERVICE
LASER OFF CANCEL/i/t');
  if key='3' then if OnEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser ON entry already
exists.  ');
     UserWait;
  end else IssueTimeCommand('on. ');
  if key='4' then if OffEntryExists then begin
     gotoxy(1,6);
     writeln('   Invalid option: Laser OFF entry already
exists. ');
     UserWait;
  end else IssueTimeCommand('off.');
until false;
end;

procedure HardReset;
begin
  port[PowerPort + 3] := $98;    { init 8255 chip }
  port[PowerPort + 3] := $98;
end;

procedure ReadParams;
begin
  assign(ParamFile,ParamFileName);
  {$I-}
  reset(ParamFile);
  IOerr := IOresult;
  if IOerr <> 0 then begin
    assign(ParamFile,HomeDirectory + ParamFileName);
    reset(ParamFile);
    IOerr := IOresult;
  end;
  {$I+}
```

```
  LaserPower := _Unknown;
  if IOerr = 0 then begin
    while not eof(ParamFile) do begin
      readln(ParamFile,ParamFileLine);
      ParamFileLine := upper(ParamFileLine);
      if (ParamFileLine <> '') and (ParamFileLine[1] <>
'!') then begin
        if copy(ParamFileLine,1,15) = 'LASERPOWERTYPE='
then begin
          temp := copy(ParamFileLine,16,3);
          if temp = 'LEV' then LaserPower := _Level
          else if temp = 'TOG' then LaserPower :=
_Toggle;
        end;
      end;
    end;
    close(ParamFile);
  end;
end;

procedure RotateStep(DirStep,InputByte: Integer);
const
  Bits: array[0..3] of Byte = ($01,$03,$02,$00);
begin
 StepIter := ((StepIter + DirStep) + 4) mod 4;
 InputByte := Bits[StepIter];
 writeln('   InputByte =',InputByte);
end;

procedure InitSweeperandPlunger;
begin
   port[$30B] := $98;
   port[$30B] := $98;
   userwait;
end;

procedure SweeperUp;
```

```
begin
   rotatestep(+1,Sweepstep);
   port[$30A] := (port[$30A] and $F3) or (sweepstep*4
and $0C);
   userwait;
end;

procedure SweeperDown;
begin
   rotatestep(-1,sweepstep);
   port[$30A] := (port[$30A] and $F3) or (sweepstep*4
and $0C);
   userwait;
end;

procedure PlungerUp;
begin
   rotatestep(+1,plungestep);
   port[$30A] := (port[$30A] and $FC) or (plungestep and
$03);
   userwait;
end;

procedure PlungerDown;
begin
   rotatestep(-1,plungestep);
   port[$30A] :=( port[$30A] and $FC) or (plungestep and
$03);
   userwait;
end;

begin
  CheckBreak := false;
  ShowStuff  := false;
  LaserOff   := false;
  SweepStep := $00;
  PlungeStep := $00;
```

```
  StepIter := 0;
  ClrScr;
  IndentStr := many(' ',Indent);
{  ReadParams; }
  CursorOff;
  repeat
    ShowPowerMenu;
    GetSelection;
    gotoxy(1,22);
    ClrEol;
    case key of
{     '1': TurnOnLaser;
      '2': TurnOnElevator;
      '3': OpenShutter;
      '4': TurnOffLaser;
      '5': TurnOffElevator;
      '6': CloseShutter;
      '7': SetLaserOnOffTime;   }
      '1': InitSweeperandPlunger;
      '2': SweeperUp;
      '3': SweeperDown;
      '4': PlungerUp;
      '5': PlungerDown;
      'X': Abort;
    end;
  until false;
  CursorOn;

end.
^Z
```

```
{============================================================
   3D Systems Stereolithography System Software
============================================================

   CALOUT.PAS
   SLA-1 Calibration Post-Processing Program

------------------------------------------------------------

   Recent History:
    3/28/88  Ver 3.00  first released for production
use.
    5/16/88  Ver 3.10  upgraded to turbo pascal version
4.0
    7/18/88  Ver 3.20  update VersionId
    9/01/88  Ver 3.21  change 'Use temperature
correction...' to
             'Adjust calibration points to correct for
measured mirror drift?'
    9/13/88  Ver 3.22  allow escape if enter is input
for filename
------------------------------------------------------------}

uses CRT,DOS;

const
   VersionId     = '3.22';
   tlo           = 0;
   thi           = 74;
   thiplus1      = 75;
   clo           = 13;
   chi           = 61;
   incint        = 889;
   CALdirectory  = '\CAL\';
   AnyFile       = $3F;

type
```

```
   str25 = string[25];
   str80 = string[80];
   arrayptr = array[tlo..thi,tlo..thi] of real;
var
   FirstVar   : byte;              { keep this in the
first location }
   DirInfo    : searchRec;
   regs       : registers;
   FileOK     : boolean;
   i,j, rangeerror  : integer;
   key  : char;
   file1, file2, file3 : text;
   DXstr, dateStr      : str25;
   InputStr, InputFile1, GEOfile, DRFfile, Astring :
str80;
   x1,y1,x2,y2 : array[tlo..thiplus1] of real;
   x3,y3       : ^arrayptr;
   x4,y4       : real;
   x3temp,y3temp : real;
   x1tot,x2tot,x3tot,x4tot,y1tot,y2tot,y3tot,y4tot :
real;
   xof,xofd,yof,yofd,xgn,xgnd,ygn,ygnd        : real;
   xd,xdmax,xdmin,yd,ydmax,ydmin              : real;
   x1dev,x2dev,y1dev,y2dev                    : real;
   ArrayIndexAlloc : Word;
   LastVar  : byte; { keep this in the last location }

{$R+,S-,I+,F-,V-,B-}

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
```

```
      Intr($10,regs);
    end;
  end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure ShowTop;
begin
   window(1,1,80,25);
   ClrScr;
   writeln;
   writeln;
   writeln('           CALOUT :  SLA-1 Calibration
Adjustment Program ');
   writeln('                     version ',versionID);
   writeln;
   writeln;
end;

function RealVal(line: string;var code1: Integer):Real;
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
      line := '-0' + line;
```

```
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code1);
   if code1 <> 0 then
val(copy(line,1,code1-1),num,code1); { was code2 }
   RealVal := num;
end;

procedure UserWait;
begin
   writeln;
   writeln('          ... press any key to continue');
   repeat delay(100) until keypressed;
   key := ReadKey;
end;


procedure PromptForCalibFile;
begin
   ShowTop;
{  exec('c:\command.com','/c dir \CAL\*.CAL');  }
   writeln('   Existing Calibration Files: ');
   write  ('       (none)    ');
   gotoxy(WhereX-16,WhereY);
   i := 0;
   FindFirst('\CAL\*.CAL',anyfile,DirInfo);
   if DosError <> 0 then begin
      writeln;
      writeln;
      UserWait;
      Halt;
   end;
   while DosError = 0 do begin
      if (i - 4) = 0 then begin
         writeln;
         i := 0;
```

```
      end;
      gotoxy(8+(15*i),WhereY);
      i := i + 1;
      write(DirInfo.Name);
      FindNext(DirInfo);
   end;
   writeln;
   gotoxy(1,WhereY);
   FileOK := false;
   repeat
      writeln;
      writeln('   Enter the calibration file in need of
adjustment:');
      CursorOn;
      readln(InputFile1);
      if InputFile1 = '' then Halt(0); {escape}
      for i := 1 to length(InputFile1) do
InputFile1[i]:=
                                   upcase(InputFile1[i]);
      i := pos('.',InputFile1);
      InputStr := InputFile1;
      if i > 0 then InputStr := copy(InputFile1,1,i-1);
      if i = 0 then InputFile1 := InputFile1 + '.CAL';
      InputFile1 := CALdirectory + InputFile1;
      InputStr := CALdirectory + InputStr;
      writeln;
      FindFirst(InputFile1,anyfile,DirInfo);
      if DosError = 0 then FileOK := true;
      if not FileOK then writeln('   Error: Invalid file
name input.');
   until FileOK;
   GEOFile := InputStr + '.GEO';
   DRFFile := InputStr + '.DRF';
end;

procedure GetCustData;
begin
```

```
   ShowTop;
   writeln('   Program to convert data from Beam
Profiler Calibration Run.');
   writeln;
   writeln('   Enter customer name and/or SLA-1 serial
number:');
   readln(InputStr);
   writeln('   Enter DX serial number:');
   readln(DXstr);
   writeln('   Enter date of calibration (mo/dy/yr):');
   readln(DateStr);
end;

procedure FillUncorrectedTable;
  var
   x3temp, y3temp : real;
begin
   ShowTop;
   writeln('   Filling Uncorrected Table
-------------------------------------------------');
   for i:= tlo to thi do begin
      x3temp := i;
      x3temp := incint * x3temp;
      if x3temp<5 then x3temp := 5;
      if x3temp>65530 then x3temp := 65530;
      for j := tlo to thi do x3^[i,j] := x3temp;
   end;
   for j:= tlo to thi do begin
      y3temp := j;
      y3temp := incint * y3temp;
      if y3temp<5 then y3temp := 5;
      if y3temp>65530 then y3temp := 65530;
      for i := tlo to thi do y3^[i,j] := y3temp;
   end;
end;

procedure FileReadError;
```

```
begin
   writeln('   ERROR encounted in reading calibration
file:');
   writeln('   CALOUT execution aborted.');
   UserWait;
   Halt(1);
end;

procedure ParseToComma(var InString: string;var xoutput
: real);
  var
   i, valcode : integer;
begin
   i := pos(',',InString);
   if i=0 then xoutput := RealVal(Instring,valcode)
   else begin
      xoutput := RealVal(copy(Instring,1,i-1),valcode);
      InString := copy(InString,i+1,length(InString));
   end;
end;

procedure ReadLineX3Y3(var x3,y3: real);
  var
   i : integer;
begin
   repeat
      readln(file1,Astring);
   until Astring[1]<>'!';
   ParseToComma(Astring,x3);
   ParseToComma(Astring,y3);
end;

procedure ReadFour(var x3,y3,x4,y4: real);
  var
   i : integer;
begin
   repeat
```

```
      readln(file1,Astring);
   until Astring[1]<>'!';
   ParseToComma(Astring,x3);
   ParseToComma(Astring,y3);
   ParseToComma(Astring,x4);
   ParseToComma(Astring,y4);
end;

procedure ReadMeasuredValues;
begin
   ShowTop;
   writeln('   Reading Measured Values
-------------------------------------------------');
   {$I-}
   assign(file1,InputFile1);
   Reset(file1);
   if (IOResult <> 0) then FileReadError;
   for i:= clo to chi do begin
      ReadFour(x1[i],y1[i],x2[i],y2[i]);
      for j := clo to chi do begin
         ReadLineX3Y3(x3^[i,j],y3^[i,j]);
         if (x3^[i,j]<8000) or (x3^[i,j]>57000) then
begin
            writeln('   WARNING: X value out of range.
X=',x3^[i,j]:6:2);
            RangeError := -1;
         end;
         if (y3^[i,j]<8000) or (y3^[i,j]>57000) then
begin
            writeln('   WARNING: Y value out of range.
Y=',y3^[i,j]:6:2);
            RangeError := -1;
         end;
      end;
   end;
   i := chi + 1;
   ReadFour(x1[i],y1[i],x2[i],y2[i]);
```

```
   close(file1);
   {$I+}
   if RangeError <> 0 then UserWait;
end;

procedure AdjustMeasuredValues;
{ NOTE: we will assume 32767 is at center of range, to
1st order
        so we can use offset & gain (of & gn)
independently

        First calculate offset & gain for each set of
readings,
        then take proportionate part of the ones for
beginning & end
        of the Y Axis (j) table movement;
        --assume drift evenly distributed over time
intervals between
          readings, remember that profile measurements
take time too.
        Xnew = ( Xold - 32767 ) / Xgn  + 32767 - Xof
        (how nice if numbers were centered on Zero!)    }


begin
   ShowTop;
   writeln('   Adjusting Measured Values
-------------------------------------------');
   for i := clo to chi do begin
   {  need the range of corrections from beginning & end
of Column }
      xof:=x1[i];                                  { start }
      xofd:=(x1[i+1]-x1[i])/(chi-clo+2);           { delta }
      yof:=y1[i];                                  { etc.. }
      yofd:=(y1[i+1]-y1[i])/(chi-clo+2);
      xgn:=x2[i];
      xgnd:=(x2[i+1]-x2[i])/(chi-clo+2);
```

```
      ygn:=y2[i];
      ygnd:=(y2[i+1]-y2[i])/(chi-clo+2);
      for j := clo to chi do begin
         x3^[i,j]:=(x3^[i,j]-32767)/(xgn+xgnd*
(j-clo+1))+32767-xof-xofd*(j-clo+1);
         y3^[i,j]:=(y3^[i,j]-32767)/(ygn+ygnd*
(j-clo+1))+32767-yof-yofd*(j-clo+1);
      end;
   end;
{ Adjusting Extrapolated Values ------------------
  do we really need to ?  - yes for 12" pot at same
level . . .
  but should be calibrating 12" by then: SO - DON'T
             ( Comment kept from the original basic
program. )     }
end;

procedure FindMirrorDrift;
begin
   ShowTop;
   writeln('   Finding Mirror Drift
-----------------------------------------------------');
   for i:= clo to chi+1 do begin
      x1tot := x1tot + x1[i]/(chi+1-clo+1);
      x2tot := x2tot + x2[i]/(chi+1-clo+1);
      y1tot := y1tot + y1[i]/(chi+1-clo+1);
      y2tot := y2tot + y2[i]/(chi+1-clo+1);
   end;
   for i:= clo to chi+1 do begin
      xd := x1tot-x1[i];
      if xdmax<xd then xdmax:=xd;
      if xdmin>xd then xdmin:=xd;
      x1dev:=x1dev+xd*xd/(chi+1-clo+1);
      xd:=x2tot-x2[i];
      if xdmax<xd then xdmax:=xd;
      if xdmin>xd then xdmin:=xd;
      x2dev:=x2dev+xd*xd/(chi+1-clo+1);
```

```
        yd:=y1tot-y1[i];
        if ydmax<yd then ydmax:=yd;
        if ydmin>yd then ydmin:=yd;
        y1dev:=y1dev+yd*yd/(chi+1-clo+1);
        yd:=y2tot-y2[i];
        if ydmax<yd then ydmax:=yd;
        if ydmin>yd then ydmin:=yd;
        y2dev:=y2dev+yd*yd/(chi+1-clo+1);
    end;
    {$I-}
    assign(file2,DRFfile);
    rewrite(file2);
  { begin }
      writeln(file2,'!CALOUT ',versionID,' Drift Report
File');
      writeln(file2,'!SLA-1 ',inputstr,' DX serial #
',dxstr,' on ',datestr);

      writeln(file2,'!Position values used for
calibration, & RMS deviations (bits)');
      write  (file2,'!Sensor1  ');
      writeln(file2,x1tot:6:2,'  ',y1tot:6:2,'
',SQRT(x1dev):6:2,'  ',SQRT(y1dev):6:2);
      write  (file2,'!Sensor2  ');
      writeln(file2,x2tot:6:2,'  ',y2tot:6:2,'
',SQRT(x2dev):6:2,'  ',SQRT(y2dev):6:2);

      writeln(file2,'   x1tot    x2tot    y1tot    y2tot
(bits)    ');
      writeln(file2,'  ',x1tot:6:2,'  ',x2tot:6:2,'
',y1tot:6:2,'  ',y2tot:6:2);
      writeln(file2,'   x1dev    x2dev    y1dev    y2dev
(bits RMS)');
      writeln(file2,'  ',SQRT(x1dev):6:2,'
',SQRT(x2dev):6:2,'  ',SQRT(y1dev):6:2,'
',SQRT(y2dev):6:2);
      writeln(file2);
```

```
      writeln(file2,'   x1[i]    x2[i]    y1[i]    y2[i]
  (bits)   ');
      for i:=clo to chi+1 do begin
         writeln(file2,'  ',
            (x1[i]-x1tot):6:2,'  ',(x2[i]-x2tot):6:2,'
',(y1[i]-y1tot):6:2,'  ',
            (y2[i]-y2tot):6:2);
      end;
 { end; }
   { re-work the drift information }
   for I:=clo to chi+1 do begin
      xof:=((x1[i]+x2[i])-(x1tot+x2tot))/2;   { bits }
      yof:=((y1[i]+y2[i])-(y1tot+y2tot))/2;   { bits }
      xgn:=(x1[i]-x2[i])/(x1tot-x2tot);       { approxi-
mately = 1 }
      ygn:=(y1[i]-y2[i])/(y1tot-y2tot);       { approxi-
mately = 1 }
    { re-use the x1() etc. arrays }
      x1[i]:=xof;
      x2[i]:=xgn;
      y1[i]:=yof;
      y2[i]:=ygn;
   end;
   writeln(file2);
   writeln(file2,'   X offset X gain   Y offset Y gain
(bits, bits/10")');
   writeln(file2);
   for i:= clo to chi+1 do begin
      writeln(file2,'  ',
      x1[i]:7:1,'  ',((x2[i]-1)*35560):7:1,'
',y1[i]:7:1,((y2[i]-1)*35560):7:1);
   end;
   close(file2);
   {$I+}
   if ydmax<yd then ydmax:=yd;
   if ydmin>yd then ydmin:=yd;
   xd := xdmax-xdmin;
```

```
   yd := ydmax-ydmin;
   if xd<yd then xd:=yd;  { select largest drift }
   writeln;
   writeln('   From measuring the apparent position of
the beam profile sensors');
   writeln('   during calibration, we find total
effective mirror drift');
   writeln('   ',round(xd/3.556),' mils ');
   UserWait;
   writeln;

   if xd>500 then begin
      writeln('   Program found wrong profiler
pinholes,');
      writeln;
      writeln('   File will be written without
attempting to correct for drift');
      UserWait;
      exit;
   end;
   if xd>35 then begin
      writeln('   WARNING: Excessive drift detected. ');
      writeln('            Possible causes are
temperature drift, ');
      writeln('            direct air currents from
heating or cooling ');
      writeln('            vents, and/or insufficient
warmup time. ');
      writeln;
      if xd>100 then begin
         writeln('   File will be written without
attempting to correct for drift');
         exit;
      end;
      writeln('   Adjust calibration points to correct
for measured mirror drift ?');
      repeat delay(100) until keypressed;
```

```
      Key := upcase(ReadKey);
      if Key = 'Y' then AdjustMeasuredValues;
   end;
end;

procedure WriteGEOFile;
begin
   ShowTop;
   writeln('   Writing Corrected File: ',GEOfile,'
----------------------------------');
   {$I-}
   assign(file3,GEOfile);
   rewrite(file3);
   writeln(file3,'!SLA-1 ',inputstr,' DX serial #
',DXStr,' on ',datestr);
   writeln(file3,'!Position values used for calibration,
& RMS deviations (bits)');
   write  (file3,'!Sensor1 ');
   writeln(file3,x1tot:6:2,'  ',y1tot:6:2,'  ',
                 SQRT(x1dev):6:2,'  ',SQRT(y1dev):6:2);
   write  (file3,'!Sensor2 ');
   writeln(file3,x2tot:6:2,'  ',y2tot:6:2,'  ',
                 SQRT(x2dev):6:2,'  ',SQRT(y2dev):6:2);
   writeln(file3,thi+1,',',thi+1,',2,',incint);

   for i := tlo to thi do begin
      for j := tlo to thi do begin
      {  str(trunc(x3^[i,j]*10+0.5)/10,Astring);
         Astring := copy(Astring,2,length(Astring));
   write  (file3,Astring,',');
         str(trunc(y3^[i,j]*10+0.5)/10,Astring);
         Astring := copy(Astring,2,length(Astring));
         writeln(file3,Astring); }
         writeln(file3,((x3^[i,j]*10+0.5)/10):7:1,',',
                       ((y3^[i,j]*10+0.5)/10):7:1);
      end;
   end;
```

```
   writeln;
   close(file3);
   {I+}
end;

procedure FinalWritelns;
begin
   ShowTop;
   writeln('   Calibration correction complete.');
   writeln;
   writeln('   Two files have been generated:
',DRFfile,' and ',GEOfile,'.');
   writeln;
   writeln('  ',DRFfile,' has the profiler positions
measured after each');
   writeln('   column of the grid.');
   writeln;
   writeln('  ',GEOfile,' has the calibration data which
will become');
   writeln('   \3DSYS\STEREO.GEO if selected during the
"install new');
   writeln('   geometric correction table" option of
program BEAM. ');
   writeln;
   writeln('   CALOUT program:  normal termination.');
   UserWait;
end;
begin { main program }
   FillChar(FirstVar,ofs(LastVar)-ofs(FirstVar),#0);
{ clear variables }
   ArrayIndexAlloc := Sqr(thi-tlo+1) * 6;
{ reals have 6 bytes }
   writeln('memavail =',MemAvail,'
Arrayindexalloc=',arrayindexalloc);
   if (MemAvail - 32768) < (2*ArrayIndexAlloc) then
begin
```

```
      writeln('   ERROR: Insufficient memory available
for CALOUT.');
      Halt(1);
   end;
   GetMem(x3,ArrayIndexAlloc);
   GetMem(y3,ArrayIndexAlloc);
   PromptForCalibFile;   { Get user calibration file. }
   GetCustData;          { Get user information such as
serial number and date. }
   FillUncorrectedTable; { Manually fill the 74 x 74
real array with defaults. }
   ReadMeasuredValues;   { Read calibration data from
input calib. file. }
   FindMirrorDrift;      { *** Find the deviation
between sensor readings. }
   WriteGEOfile;
     { Build corrected .GEO file by altering the input
from the calibration
       file. }
   FinalWritelns;        { Closing proceedings. }
end.

{  ADDITIONAL NOTES:
     1. The calibration table has a sensor every 889
bits ( 0.2500").

     2. The nearest sensor above 65536 is 74 x 889 =
65786 = 65535 + 251, thus our table goes from 0 to 74.

     3. The nearest reading to center is 37 x 889 =
32767 + 126 (0.0354"), thus our readings (-6 to 6") are
from array locations 13 to 61.

     4. The X-axis is reversed, it is corrected by
reading X from 6 to -6.
```

```
     5. Block zeros over variable declaration emulates
BASIC setting all variables to zero as they are
introduced.

     6. Pointers were required due to PASCAL limiting
the number of variable definitions.          ********* )

{$R-,S+,I+,D+,T-,F-,V+,B-,N-,L+ }
{$M 4096,0,0 }
{ SLA-1 Release 3.0X Installation Program  }

{*********

  4/00/88 ver1    r306  written this date
  7/18/88 ver2    r306  modified to r320
  7/21/88 ver21   correct NMI disable from port[305] to
port[$305]
  7/22/88 ver320  deleted writlns on super.prm
  7/25/88 ver320  add rmdir CAL in erase everything
  7/28/88 ver320  add option to format a diskette before
archive
  8/22/23 ver321  add ipconfig ipcust.sys hostname
process
  8/22/23 ver321  rename autoexec and config to
autoexec.sav, config.sav
  8/25/88 ver322  add stereo.def to update, add
installation ver #
 ********************************** end NOTES }

Program Install;
Uses
    Crt, Dos;
Const
     UpBytes = 1024000;
     NewBytes = 20000000;
Var
   CurDir      : String[80];
```

```
   BytesFree    : Longint;
   InChar, key  : Char;
   regs         : registers;
   fn, fa, fc   : text;
   fileexists   : boolean;
   month,day,year,dayofweek : word;

Procedure DosCmnd(Command: string);
Begin
     writeln(Command);
     Exec('c:\command.com', Concat('/c ',Command));
     If ( DosError <> 0 ) then Halt;
     writeln;
End;

Procedure DosCmndNoHalt(Command: string);
Begin
     writeln(Command);
     Exec('c:\command.com', Concat('/c ',Command));
     If ( DosError <> 0 ) then writeln('   DOS return
code=',DosError);
     writeln;
End;

procedure ShowTop;
begin
   window(1,1,80,25);
   ClrScr;
   GotoXY(1,2);
   writeln('        Release 3.20  Stereolithography
Installation Program  ');
   writeln('                      Installation Version
3.22               ');
   Window(1,5,80,25);
end;

procedure UserWait;
```

```
begin
   writeln;
   writeln;
   writeln('              ... press any key when ready
to continue     ');
   repeat Delay(200) until Keypressed;
   key := Readkey;
   writeln;
end;

procedure ForceReBoot;
begin
     window(1,1,80,25);
     ClrScr;
     writeln;
     DosCmnd('type c:\3DSYS\read.me');
     writeln('              ... press any key when ready
to continue     ');
     repeat Delay(200) until Keypressed;
     key := Readkey;
     ShowTop;
     writeln;
     Writeln('           SLA R3.20 Installation
Completed.      ');
     writeln;
     writeln;
     writeln;
     writeln;
     writeln;
     writeln('      *****  PLEASE REMOVE THE FLOPPY DISK
IN DRIVE "A"    ');
     writeln;
     writeln;
     writeln('        When complete, press any key to
run the new ');
     writeln('          release 3.20 Stereolithography
Software :  ');
```

```
      writeln;
      UserWait;
    { Intr($19,regs); }
      port[$305] := port[$305] or 7;
      Inline($B8/$40/$00/$8E/$D8/$BB/$72/$00/$C7/$07/$34/
$00/$C7/$47/$01/$12/$00/
           $EA/$00/$00/$FF/$FF/
           $01/$04/$43/$4F/$50/$59/$03/$61/$26/$05/$50/
$41);
end;

procedure RecordDate;
begin
     ShowTop;
     {$I-}
     DosCmndNoHalt('mkdir c:\HISTORY');
     assign(fn,'c:\HISTORY\Maint.log');
     reset(fn);
     fileexists := (IOResult = 0);
     if not fileexists then rewrite(fn);
     append(fn);
     GetDate(Year,month,day,dayofweek);
     writeln(fn);
     if fileexists then write(fn,'Added to this file:')
                   else write(fn,'Started this file:');
     writeln(fn,month,'/',day,'/',year);
     write(fn,'   ACTION:');
     Case InChar of
       '1' : writeln(fn,' Installed new R3.20 SLA
software.');
       '2' : writeln(fn,' Updated R3.06 to R3.20 SLA
software.');
     end;
     close(fn);
end; { record date }

procedure ReLoadR320diskette;
```

```
begin
     ShowTop;
     writeln('   ***** Please reload the original R320
SLA installation diskette:');
     writeln('   ( SLA 3.20    Volume 1 of 1 )');
     UserWait;
end;

procedure ArchR306;
begin
     ShowTop;
     writeln('   ***** Please insert a high density
diskette for archiving R3.06:');
     writeln;
     writeln(' ... press  B  to bypass the archive');
     writeln(' or  Q  to quit the program');
     writeln(' or  F  to format a diskette and receive
archive:');
     writeln(' or  A  when a previously formated
diskette ');
     writeln(' is loaded and ready to receive
archive: ');
     repeat delay(100) until KeyPressed;
     key := upcase(readkey);
     if key = 'B' then begin
        writeln;
        writeln;
        writeln('  This is your last chance to archive
R3.06...   ');
        writeln('     ARE YOU SURE YOU WANT TO SKIP THE
ARCHIEVE ?');
        writeln;
        repeat delay(100) until KeyPressed;
        key := upcase(ReadKey);
        if key='Y' then key:='B' else key:='A';
     end;
     if  key =  'F' then DosCmnd('format a:');
```

```
        if  key =  'Q' then Halt;
        if  key <> 'B' then begin
           DosCmndNoHalt('mkdir a:\R306SLA');
           DosCmnd('copy c:\3DSYS\*.* a:\R306SLA');
           ReLoadR320diskette;
        end;
        writeln;
        DosCmnd('erase c:\3DSYS\*.exe');
end;

Procedure CopyQDOS;
begin
        ShowTop;
        writeln('   ***** Please load the QDos diskette:');
        writeln('   ( Q-DOSII by Gazelle Systems )');
        writeln;
        UserWait;
        { QDSTART makes its own QDOS directory. }
        DosCmnd('a:QDSTART');
        writeln;
        writeln;
        writeln('   QDOS II successfully loaded onto
harddisk.');
end;

procedure CopyFTP;
begin
        ShowTop;
        writeln;
        writeln('   ***** Please load the FTP diskette
10:');
        writeln(' ( MICOM- Interlan, Inc.  VOL X of X )');
        UserWait;
        DosCmnd('copy a:\FTPSRV.exe c:\FTP');
        ShowTop;
        writeln;
        writeln(' ***** Please load the FTP diskette 6:');
```

```
        writeln(' ( MICOM- Interlan, Inc.  VOL VI of X )');
        UserWait;
        writeln;
        DosCmnd('copy a:\FTP.exe c:\FTP');
        DosCmnd('copy a:\IFCONFIG.exe c:\FTP');
        DosCmnd('copy a:\IPCONFIG.exe c:\FTP');
        DosCmnd('copy a:\IPCUST.sys c:\FTP');
        DosCmnd('copy a:\NI5210.sys c:\FTP');
        DosCmnd('copy a:\TNVT100.exe c:\FTP\TELNET.exe');
        DosCmnd('\FTP\iPconfig \FTP\iPcust.sys hosttable
c:\ftp\host.txt');
        DosCmnd('\FTP\iPconfig \FTP\iPcust.sys bs');

        DosCmnd('\FTP\iFconfig \FTP\NI5210.sys 89.0.3.1');
        DosCmnd('\FTP\iFconfig \FTP\NI5210.sys Memory
D800');
        DosCmnd('ipconfig ipcust.sys hostname Process');
        writeln;
        writeln('   FTP successfully loaded onto
harddisk.');
end;

procedure CopyCAL;
begin
        DosCmndNoHalt('mkdir c:\CAL');
        DosCmndNoHalt('copy c:\3DSYS\*.cal c:\CAL\*.cal');
        DosCmndNoHalt('copy c:\WORK\*.cal c:\CAL\*.cal');
        DosCmndNoHalt('copy a:\CAL\*.geo c:\CAL\*.geo');
{ STEREO.GEO }
end;

procedure CopyR320Root;
begin
        ShowTop;
        Assign (fa,'c:\autoexec.bat');
        fileexists := (IOResult = 0);
        if fileexists then Rename (fa,'autoexec.sav');
```

```
      Assign (fc,'c:\config.sys');
      fileexists := (IOResult = 0);
      if fileexists then Rename (fc,'config.sav');
      DosCmnd('copy a:\Rootx\*.* c:\');
end;

procedure CopyR320Updates;
begin
   ShowTop;
   DosCmnd('copy a:\3DSYS\*.exe c:\3DSYS');
   DosCmnd('copy a:\3DSYS\stereo.def c:\3DSYS');
   DosCmnd('copy a:\3DSYS\read.me c:\3DSYS');
   DosCmnd('copy a:\rootx\chrono.sys c:\');
   writeln;
   CopyCAL;
   writeln;
end;

procedure CopyR320New;
begin
      ShowTop;
      DosCmnd('mkdir c:\3DSYS');
      writeln;
      DosCmnd('copy a:\3DSYS\*.* c:\3DSYS');
      DosCmnd('copy a:\3DSYS\*.pr^ c:\3DSYS\*.prm');
      DosCmnd('mkdir c:\3DSYS\EDITOR');
      DosCmnd('copy a:\3DSYS\Editor\*.*
c:\3DSYS\EDITOR');
      DosCmnd('mkdir c:\WORK');
      DosCmnd('copy a:\WORK\Sample.Mat c:\WORK');
      DosCmnd('mkdir c:\HISTORY');
      DosCmnd('mkdir c:\FIELD');
      DosCmnd('copy a:\Field\*.* c:\Field');
      DosCmnd('mkdir c:\FTP');
      DosCmnd('mkdir c:\CAL');
      DosCmnd('copy a:\FTP\Host.txt c:\FTP');
      writeln;
```

```
     writeln;
     writeln('   Primary SLA system successfully loaded
onto harddisk.')
end;

procedure ExplainUpdateR306toR320;
begin
   ShowTop;
   writeln;
   writeln('                Update release 3.06 to
release 3.20  ');
   writeln('                    INFORMATION PANEL   ');
   writeln;
   writeln;
   writeln('   The first step in the update process is
to archive the release 3.20');
   writeln(' files residing in the \3DSYS directory. ');
   writeln;
   writeln('   Next, the SLA programs in the \3DSYS
directory are updated to the ');
   Writeln('   release 3.20 level.');
   writeln;
   writeln('   The .EXE files in \3DSYS will be deleted.
None of the .PRM files in');
   Writeln('   the \3DSYS or \WORK directory will be
modified.');
   Writeln;
   UserWait;
end;

procedure NewR320SLA;
(*** Perhaps this procedure should verify that no SLA
software exists.***)
begin
     CopyR320Root;
     CopyR320New;
     CopyQDOS;
```

```
        CopyFTP;
        RecordDate;
        ForceReBoot;
end;

Procedure UpdateR306toR320;
Begin
        ShowTop;
        If ( UpBytes = BytesFree ) Then
        Begin

writeln('*******************************************');
                writeln;
                Writeln('        The SLA R320 software update
requires ', UpBytes, ' Bytes');
                Writeln('             but only ', BytesFree, '
are currently available.');
                writeln;
                writeln;
                writeln;
                writeln('          Please clear enough memory
space on the hard disk');
                writeln(' and re-try the SLA software
update.');
                writeln;
                writeln;
                writeln;
                Writeln('         Execution Terminated. ');
                Halt;
        End;
        ExplainUpdateR306toR320;
        ArchR306;
        CopyR320Updates;
        RecordDate;
        ForceReBoot;
End;
```

```
procedure ShowInstallMenu;
begin
     Window(1,1,80,25);
     ClrScr;
     gotoxy(9,8);
     Writeln('*******  SLA-1 Installation Program
Primary Menu *******');
     Writeln;
     Writeln(' 1) Install a new R320 SLA System');
     Writeln(' 2) Update R306 to R320 SLA Software');
     Writeln(' 3) Information Panel ');
     Writeln;
     Writeln(' Q) Quit Installation Program');
     Writeln;
     Write  (' Enter Option:');
end;

procedure GiveHelpInfo;
begin
   ShowTop;
   writeln;
   writeln('   *****  Information Panel  *****');
   writeln;
   writeln;
   writeln('     This software allows fast, easy
installation of 3D System''s');
   writeln('     control computer enviornment.   ');
   writeln;
   writeln('     A new installation of R3.20 requires a
QDOS II installation disk');
   writeln('     and the MICOM Interlan (FTP)
installation disks.');
   writeln;
   writeln('     For an update from R3.06 to R3.20 use a
formatted high density');
   writeln('     floppy disk for archiving the R3.06
software.  This is not mandatory,');
```

```
   writeln('     only suggested. ');
   writeln;
   writeln('     All customized parameter files (*.PRM)
should be kept in the \WORK');
   writeln('     directory.  ');
   writeln;
   writeln('     If you wish to install a color monitor
please contact 3D systems.');
   UserWait;
end;

procedure EraseEverything;
begin
   ShowTop;
   writeln(' ==============>  CRITICAL WARNING
<==========');
   writeln;
   writeln('    THIS IS OPTION ERASE EVERYTHING:     ');
   writeln;
   writeln;
   writeln;
   writeln(' ... press  E  to erase everything except');
   writeln('       the DOS directory and COMMAND.COM ');
   writeln;
   writeln(' press any other key to leave this option');
   writeln;
   repeat delay(100) until KeyPressed;
   key := upcase(readkey);
   if  key <> 'E' then exit;
   writeln;
   DosCmndNoHalt('c:');
   DosCmndNoHalt('erase c:\3DSYS\EDITOR\*.*');
   DosCmndNoHalt('cd c:\3DSYS');
   DosCmndNoHalt('rmdir EDITOR');

   DosCmndNoHalt('erase c:\3DSYS\*.*');
   DosCmndNoHalt('cd c:\');
```

```
    DosCmndNoHalt('rmdir 3DSYS');

    DosCmndNoHalt('erase c:\FIELD\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir FIELD');

    DosCmndNoHalt('erase c:\WORK\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir WORK');

    DosCmndNoHalt('erase c:\HISTORY\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir HISTORY');

    DosCmndNoHalt('erase c:\QDOS\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir QDOS');

    DosCmndNoHalt('erase c:\FTP\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir FTP');

    DosCmndNoHalt('erase c:\CAL\*.*');
    DosCmndNoHalt('cd c:\');
    DosCmndNoHalt('rmdir CAL');

    DosCmndNoHalt('erase c:\*.sys');
    DosCmndNoHalt('erase c:\*.bat');
    writeln;
    writeln('   Everything is erased.   ');
    writeln;
    UserWait;
end;

Begin
     ClrScr;
     BytesFree := DiskFree(3);
```

```
      DOScmnd('verify on');
      repeat
         ShowInstallMenu;
         InChar := 'Z';
         repeat
            repeat delay(100) until Keypressed;
            InChar:= upcase(ReadKey);
         until InChar in ['1','2','3','Q','Z'];
         Case InChar of
           '1': NewR320SLA;        { Install everything
from ground zero. }
           '2': UpdateR306toR320;  { This option archives
the old version software and doesn't need qdos, et al.,
installed. }
           '3': GiveHelpInfo;      { Passive help file. }
           'Q': exit;
           'Z': EraseEverything
         end;
       until false;
End.

^Z
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================

   MERGE.PAS
   SLA-1 Merge Sliced Files Software

---------------------------------------------------------
   Compile with Turbo Pascal Version 4.0

   File name line syntax:

   <file name>/Z<offset>/X<offset>/Z<offset>
                          /N<blocks>/O<blocks> <file
name>...

/OUTFILE<outfile>/RES<resolution>/ZSPC<z-spacing>/THICK<
thickness>

   where <file name> is a Slice input file
         <offset>    is a + or -  X,Y or Z-axis offset
for that input file
         <blocks>    contains the identifiers for the
blocks to skip
                     for that input file separated by
commas
         <outfile>   follows all input file specs and is
the name of
                     the part output file to store the
parsed data

   Up to 10 Slice data files can be combined.
---------------------------------------------------------

   Merge *.V vector output file format:    (file #2)

     L1,<layer A>
```

```
     <x1> <y1> <x2> <y2>
      :    :
     <another block,<layer A>
      :    :
    [L2,<layer A>
     <x1> <y1> <x2> <y2>
      :    :
     <another block>,<layer A>
      :    :  ]
     L1,<layer B>
      :    :
     (EOF)
```

Critical Area Codes:

XV = draw section cross-hatch vectors in box as rivets
XI = ignore section cross-hatch vectors in box
SV = draw up/down skin cross-hatch vectors in box as rivets
SI = ignore up/down skin cross-hatch vectors in box

Recent History:

9/ 1/87	Ver 2.04	Version 2 released
9/ 3/87	Ver 2.07	Compiled by TBM (Modified Turbo Basic)
9/ 4/87	Ver 2.08	added SV, SI critical areas
9/17/87	Ver 2.10	only updated version number
9/25/87	Ver 2.20	new SLICE 19 block format supported
9/28/87	Ver 2.21	added NFUB support
9/30/87	Ver 2.25	only updated version number

```
   10/26/87  Ver 2.27  Parse name changed to Merge
           *.P files now *.V files
           error file support
   11/12/87  Ver 2.30  first Beta software release
   11/20/87  Ver 2.31  added directory listing of *.SLI
files
             Ver 2.32  -
   12/04/87  Ver 2.33  slice20 FUB FDB blocks
           BLOCK.ID file support
   12/14/87  Ver 2.40  input file names in *.L output
file
           pass along ! comment lines in slice files
   12/21/87  Ver 2.41  second !MERGE line contains
start, end, # of layers
           BOX file noted when found, not if not found
           scans !SLICE comments for Res, ZSpacing,
Thick
    1/11/88  Ver 2.50  BLOCK.ID looked for in \3DSYS too

    1/22/88  Ver 2.54  makes Range instead of Layer
Control File as default
                       Layer Control File made if
/LAYERCTRL option used
    1/25/88  Ver 2.60  generates both Layer and Range
Files (.L and .R)
    1/27/88            thickness input is in mils, not
millimeters
    1/27/88            thickness defaults to Z-Spacing
value
    2/10/88            converted from Turbo Basic to
Turbo Pascal 4.0
                       merge operation runs 2.5 times
faster than before
                       merge status box with layer and
range information
                       /O only and /N no block
selections improved
```

```
    2/12/88  Ver 2.61  EndLayer variable now LongInt
                       layer numeric output is type
word, not integer
    3/09/88  Ver 2.80  added copyright notice
    3/22/88  Ver 2.90  Create binary V file
                       Add file offsetting option
    4/08/88  Ver 2.91  Eliminate Critical Area Boxes
                       Add Min Max values to Headers
    4/09/88  Ver 3.00  Production
    4/15/88  Ver 3.01  Replace /Offset#,# (X and Y) with
/X# and /Y# to
                       parallel the /Z# option; also set
RC<>0 for case
                       of merge file not found.
    4/22/88  Ver 3.02  better error checking input
numbers from SLI file
                       correct exit procedure to pass
errors
                       correct ASCII reading only one
file
    6/14/88  Ver 3.10  correct disk full error
    6/22/88  Ver 3.11  add automatic centering and
offsets to partname.PRM
                       ( formerly BUILD.PRM )
    6/30/88  Ver 3.12  use 3556/SliceResolution instead
of XYscale in
                       Build.PRM for the scale when
determining offsets and
                       build windowing
    7/05/88  Ver 3.13  also update partname.prm
XYonly-scale-factor or the
                       other updates are worthless
    7/18/88  Ver 3.20  update VersionId
    9/16/88  Ver 3.21  fix numlayers +1 problem if num
ranges > 1, and Scott's
-------------------------------------------------------}
```

```
{$R-,S-,I+,D-,T-,F-,V-,B-,N-,L+ }
{$M 16384,0,655360 }
{$IFOPT R+}
  {$DEFINE RNGON}
{$ELSE}
  {$UNDEF RNGON}
{$ENDIF}

uses Crt,Dos;

const
  VersionId = '3.21';
  sp80 = '                                        ';
  ErrorFileName = 'C:\SYSTEM.ERR';
  BlockIdFileName = 'BLOCK.ID';
  HomeDirectory = 'C:\3DSYS\';
  Cr = ^M;
  StatusBoxLeft   = 42;
  StatusBoxRight  = 79;
  StatusBoxTop    = 10;
  StatusBoxBottom = 22;

type
  str6  = string[6];
  str16 = string[16];

var
  FirstVar: Byte;  { must be first variable defined }
  ExitSave : Pointer;
  savedx,savedy,saveRLC,quikfixcnt: integer;
   { for correcting V3.20 to V3.21 rangelayercount }
  CurrentZ,ZOffset: array [1..10] of LongInt;
  OnlyBlocks,NoBlocks: array [1..10] of String;
  BlockMnemonics: array [1..32] of str6;
  Blocks: array [1..100] of str6;
  ErrorFile: Text;
  InputFiles: array[1..10] of Text;
```

```
  code,NumPartFiles,InFile,NumCritVols: Integer;
  BlockMnemonicsCount,ZSpace,BlockCount,Layers: Integer;
  CmdLine,MergeLine,FirstFileName,Line,OutFileName:
String;
  SingleBar,LayerFileName,RangeFileName,VectorFileName:
String;
  Resolution,ZSpacing,Thickness: Real;
  CurrentLayer,LastLayer,ZStart,ZEnd,ZVal: LongInt;
  StartLayer,EndLayer,MergeLayer: LongInt;
  LayerFile,RangeFile,VectorTextFile: Text;
  VectorBinaryFile : File;
  BinaryOutput : Boolean;
  UpdateFile: File of Char;
  UpdateIndex: LongInt;
  HeaderType: Char;
  FileNum,Cmd,Blk: str6;
  FirstRange,NewRange,SkipBlock,NeedStep: Boolean;
  RangeLayerCount,RangeCount,UpdX,UpdY: Integer;
  OffsetX,OffsetY : Array [1..10] of LongInt;
  VectorCount : LongInt;
  VectorCountPtr : LongInt;
  rc : Integer;
  MinX,MaxX,MinY,MaxY : Word;
  LastVar: Byte;  { should be last variable defined }


{-------------------------------------------------------

                              Utility Routines

-------------------------------------------------------}

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
```

```
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;
```

```
function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
  Str(num,temp);
  LongStr := temp;
end;

function RealStr(num: Real;AfterDecimal: Integer):
String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal,temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc := copy(sp80+sp80,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
```

```
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;

{
  Find any char of Chars in Main starting at index Start

  Returns length(Main)+1 if no character of Chars in
Main
}
function FindStr(Start: Integer;Main: string;Chars:
str16): Integer;
var
  i,j,min: Integer;
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
```

```
  for i:=1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  {$R-}
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  {$IFDEF RNGON}
    {$R+}
  {$ENDIF}
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
```

```
var
  ofs,code2: Integer;
  num: Real;
begin
  ofs := 0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end;
  {$R-}
  val(line,num,code);
  if code<>0 then begin
    val(copy(line,1,code-1),num,code2);
    inc(code,ofs);
  end;
  {$IFDEF RNGON}
    {$R+}
  {$ENDIF}
  RealVal := num;
end;

{
  Pick out a number from a string
  starting at Index and going to the next space or comma
or end-of-string
}
procedure UnitValShortInt(var Storage: Integer;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;
```

```
procedure UnitValInt(var Storage: LongInt;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := LongVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure UnitVal(var Storage: Real;Text: String;var
Index: Integer);
var
  i: Integer;
begin
  Storage := RealVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

{
  Pick out text from a string
  starting at Index and going to the next comma or
end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
Integer);
var
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

{
  Show error and store it on disk
```

```
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'MERGE: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{ Harry's ultimate Draw-A-Box on the text screen
routine }
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
```

```
                { Type          1    2    3    4     }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then begin
    if y1=y2 then FullHorzBar := Many(HorzBar[BoxType],
x2-x1+1)
      else FullHorzBar := Many(HorzBar[BoxType],
x2-x1-1);
    end else FullHorzBar := '';
  if y1<>y2 then begin
    if x1<>x2 then write(UpperLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(UpperRightCorner[BoxType]);
  end;
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
  if y1 = y2 then write(FullHorzBar)
  else begin
    if x1<>x2 then write(LowerLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(LowerRightCorner[BoxType]);
  end;
end;
```

```
procedure SetUpStatusBox;
begin
  DrawBox(StatusBoxLeft,StatusBoxTop,StatusBoxRight,
StatusBoxBottom,1);
  DrawBox(StatusBoxLeft+1,StatusBoxTop+2,
StatusBoxRight-1,StatusBoxTop+2,1);
  DrawBox(StatusBoxLeft+1,StatusBoxBottom-2,
StatusBoxRight-1,StatusBoxBottom-2,1);
end;

{$F+}
procedure ExitMerge;
{$F-}
begin
  window(1,1,80,25);
  gotoxy(1,22);
  if ExitCode=101 then begin
    writeln('Error: Out of Disk Space');
    ErrorAddr:=Nil;
  end;
  ExitProc := ExitSave;
end;


{=======================================================}

procedure FindFirstLayers;
var
  i,dummy: Integer;
  GoodLine: Boolean;
begin
{ determine first layer number for each slice file }
  MergeLayer := 99999;
  for InFile := 1 to NumPartFiles do begin
    if not eof(InputFiles[InFile]) then begin
```

```
      GoodLine := false;
      repeat
        readln(InputFiles[InFile],Line);
        if (Line = '') or (Line[1] in ['!','*']) then
begin
          writeln(LayerFile,Line);
          writeln(RangeFile,Line);
          if copy(Line,1,6) = '!SLICE' then begin
            i := pos('-RES ',Line);
            if (i>0) and (Resolution<=0) then
                Resolution :=
RealVal(copy(Line,i+5,99),dummy);
            i := pos('-ZS ',Line);
            if (i>0) and (ZSpacing<=0) then
              ZSpacing :=
RealVal(copy(Line,i+4,99),dummy);
            i := pos('-THICK ',Line);
            if (i>0) and (Thickness<=0) then
              Thickness :=
RealVal(copy(Line,i+7,99),dummy);
          end;
        end else GoodLine := true;
      until GoodLine or eof(InputFiles[InFile]);
      HeaderType := Line[1];
      Zval := LongVal(copy(Line,3,99),dummy) +
ZOffset[InFile];
      CurrentZ[InFile] := Zval;
      if Zval < MergeLayer THEN MergeLayer := Zval;
    end;
  end;
  StartLayer := MergeLayer;
end;

procedure GetParameters;
var
  MmFlag: Boolean;
  dummy: Integer;
```

```
  DefaultThickness: Real;
  temp: String;
begin
  writeln;
  if Resolution <= 0 then repeat
    write('Slice -res resolution value? ');
    readln(Line);
    Resolution := RealVal(Line,dummy);
  until Resolution>0;
  if ZSpacing <= 0 then begin
    write('Slice -zs layer spacing? ');
    readln(Line);
    ZSpacing := RealVal(Line,dummy);
  end;
  MmFlag := false;
  if Thickness  <= 0 then begin
    DefaultThickness := int(ZSpacing * 1000 /
Resolution);
{    write('Layer thickness in mils
[',RealStr(DefaultThickness,0),']? ');
    readln(Line);
    if Line = '' then Thickness := DefaultThickness
    else begin
      if pos('MM',upper(Line))>0 then MmFlag := true;
      Thickness := RealVal(Line,dummy);
    end;  oooo, Scott's gonna get hell from the old
farts for this.  }
  end;
  Thickness := DefaultThickness;
  if not MmFlag then Thickness := Thickness * 0.0254;
{ convert to mm }
  temp := '* ' + RealStr(Resolution,3) + ',' +
RealStr(ZSpacing,3) + ','
    + RealStr(Thickness,3);
  writeln(LayerFile,temp);
  writeln(RangeFile,temp);
end;
```

```
{-------------------------------------------------}

procedure AddBlock;
var
  i: Integer;
begin
  i := 1;
  while (i <= BlockCount) and (Blocks[i] <> Blk) do
inc(i);
  if i > BlockCount then begin
    inc(BlockCount);
    Blocks[BlockCount] := Blk;
  end;
end;

procedure OutputRange;
var
  i: Integer;
begin
{} gotoxy(UpdX,UpdY); { save locations every time in
case its the last }
   inc(quikfixcnt);
   savedx := UpdX+6;
   savedy := UpdY;
   if quikfixcnt = 1 then saveRLC := RangeLayerCount-1;
  write(LastLayer:5,' ',RangeLayerCount:5);
  NewRange := true;
  writeln(RangeFile,ZStart,',',LastLayer,',',ZSpace,'
,',ZSpace);
   for i:=1 to BlockCount do
     writeln(RangeFile,Blocks[i]);
 inc(RangeCount);
end;

procedure StartNewRange;
begin
```

```
  if (quikfixcnt = 1) and (RangeCount = 1) then begin
        gotoxy(savedx,savedy);
        write(saveRLC:5);  { RangeLayerCount - 1 }
        gotoxy(UpdX,UpdY);
    {} end;
  ZStart := MergeLayer;
  BlockCount := 0;
  NeedStep := true;
end;

procedure MakeControl;
var
  LayerDiff: LongInt;
begin
  writeln(LayerFile,MergeLayer,',',Blk);
  if Layers=1 then begin
    ZStart := MergeLayer;
    CurrentLayer := MergeLayer;
    NeedStep := true;
    AddBlock;
  end else begin
    if CurrentLayer = MergeLayer then AddBlock
    else begin
      CurrentLayer := MergeLayer;
      if NeedStep then begin
        NeedStep := false;
        ZSpace := MergeLayer - LastLayer;
        AddBlock;
      end else begin
        LayerDiff := MergeLayer - LastLayer;
        if LayerDiff <> ZSpace then begin
          OutputRange;
          StartNewRange;
          AddBlock;
        end else AddBlock;
      end;
    end;
```

```
  end;
end;

{-------------------------------------------------}

procedure NewLayer;
var
  i: Integer;
begin
  inc(Layers);
  EndLayer := MergeLayer;
  if NewRange then begin
    if FirstRange then begin
      LastLayer := MergeLayer;
      FirstRange := false;
    end;
    writeln;
    write(Cr,RangeCount+1:2,'. ',LastLayer:5,' - ');
    UpdX := whereX;
    UpdY := whereY;
    write(LastLayer:5,'     1 layers');
    NewRange := false;
    RangeLayerCount := 1;
  end else begin
    gotoxy(UpdX,UpdY);
    inc(RangeLayerCount);
    write(LastLayer:5,' ',RangeLayerCount:5);
  end;
  Blk := '#TOP';
  MakeControl;
end;

{-------------------------------------------------}

procedure OutBinBlock(BlockStr : String);
  var
    i : Integer;
```

```
    BlockChar : Char;
  begin
    i:=1;
    while (i<=BlockMnemonicsCount) and
(BlockMnemonics[i]<>BlockStr) do Inc(i);
    if i>BlockMnemonicsCount then BlockChar := #$FF
    else BlockChar := Char (i);
    BlockWrite(VectorBinaryFile,BlockChar,1);
  end;

procedure OutBinSet(MergeSetStr : String);
  var
    MergeSetNum : Byte;
  begin
    if Length(MergeSetStr)>2 then rc:=-1
    else Val(MergeSetStr,MergeSetNum,rc);
    if rc<>0 then MergeSetNum := $FF;
    BlockWrite(VectorBinaryFile,MergeSetNum,1);
  end;

procedure OutBinNum(LongNum : LongInt);
  var
    WordNum : Word;
  begin
    if LongNum>65535 then LongNum := $FFFF;
    WordNum := LongNum;
    BlockWrite(VectorBinaryFile,WordNum,2);
  end;

procedure OutVectorLine(line : String;FileNum :
Integer);
  var
    x,y,i : Integer;
    TempWord : Word;
    TempLong : LongInt;
  begin
    x:=1;
```

```
    for i:=1 to 4 do begin
      while (x<=Length(line)) and (line[x]=' ') do
Inc(x);
      if x>Length(Line) then begin
        writeln;
        SystemError('Error: point not found in merge
file');
        writeln('File ',FileNum,' "',line,'"');
        Halt(1);
      end;
      y:=Pos(' ',Copy(line,x,Length(line)-x+1));
      if y=0 then y := Length(line)+1
      else y:=y+x-1;
      {$R-}
      Val(Copy(line,x,y-x),TempLong,rc);
      {$IFDEF RNGON}
        {$R+}
      {$ENDIF}
      if (rc<>0) or (TempLong<0) or (TempLong>65535)
then begin
        writeln;
        SystemError('Error in point in merge file
line');
        writeln('File ',FileNum,' "',line,'"');
        Halt(1);
      end;
      TempWord := TempLong;
      x:=y;
      if (i=1) or (i=3) then begin
        TempWord:=TempWord+OffsetX[FileNum];
        if TempWord<MinX then MinX:=TempWord;
        if TempWord>MaxX then MaxX:=TempWord;
      end else begin
        TempWord:=TempWord+OffsetY[FileNum];
        if TempWord<MinY then MinY:=TempWord;
        if TempWord>MaxY then MaxY:=TempWord;
      end;
```

```
      if BinaryOutput then
BlockWrite(VectorBinaryFile,TempWord,2)
      else begin
        if i=1 then write(VectorTextFile,TempWord)
        else write(VectorTextFile,' ',TempWord);
      end;
    end;
    if Not BinaryOutput then writeln(VectorTextFile);
    Inc(VectorCount);
  end;


procedure WriteVectorCount;
  begin
    if VectorCount>0 then begin
      seek(VectorBinaryFile,VectorCountPtr);
      BlockWrite(VectorBinaryFile,VectorCount,4);
      VectorCount := 0;
      seek(VectorBinaryFile,FileSize(VectorBinaryFile));

    end;
  end;

{-------------------------------------------------}

procedure PrintHeaders;
begin
(*   write(Cmd,FileNum,' '); *)
  { update vector file }
  if BinaryOutput then begin
    WriteVectorCount;
    OutBinBlock(Cmd);
    OutBinSet(FileNum);
    OutBinNum(MergeLayer);
    VectorCountPtr := FileSize(VectorBinaryFile);
    BlockWrite(VectorBinaryFile,VectorCount,4);
```

```
  end else
writeln(VectorTextFile,Cmd,FileNum,',',MergeLayer);
  { update layer and range files }
  Blk := Cmd + FileNum;
  MakeControl;
end;

{-------------------------------------------------}

procedure ReadSlices;
var
  i,j,c: Integer;
  select,temp: String;
begin
  FirstRange := true;
  NewRange   := true;
  repeat
    NewLayer;
    for InFile := 1 to NumPartFiles do begin
      FileNum := IntStr(InFile);
      { process rest of Z block & other blocks }
      SkipBlock := false;
      if not(eof(InputFiles[InFile])) and
(CurrentZ[InFile]=MergeLayer) then begin
        Cmd := HeaderType;
        PrintHeaders;
      end;
      while not(eof(InputFiles[InFile])) and
(CurrentZ[InFile]=MergeLayer) do begin
        repeat
          readln(InputFiles[InFile],Line);
          Line[length(Line) + 1] := #0;
          if copy(Line,1,1) = '!' then begin
            writeln(LayerFile,Line);
            writeln(RangeFile,Line);
          end;
        until copy(Line,1,1)<>'!';
```

```
{
  process slice block
}
        i := 0;
        temp := copy(Line,1,2);
        if (temp = 'Z ') or (temp = 'L ') then i := 1
        else if length(temp)=1 then i := pos(temp,
'ZXYISHVABJCDEFGK');
        if i=0 then begin
          i := 1;
          while (i <= BlockMnemonicsCount) and
            ((copy(Line,1,length(BlockMnemonics[i])) <>
BlockMnemonics[i])
              or not (Line[length(BlockMnemonics[i])+1]
in [#0,' ']))
                do inc(i);
          if i > BlockMnemonicsCount then i := 0;
        end;
        if i > 0 then begin
          SkipBlock := false;
          c := FindStr(1,Line,' ');
          Cmd := copy(Line,1,c-1);
          if i=1 then begin { handle Z block special }
            Zval := LongVal(copy(Line,c+1,9),code) +
ZOffset[InFile];
            CurrentZ[InFile] := Zval;
          end else begin
            select := ',' + cmd + ',';
            if (pos(select,NoBlocks[InFile]) > 0) or
              ((length(OnlyBlocks[InFile]) > 0) and
                (pos(select,OnlyBlocks[InFile])=0)) then
begin
              SkipBlock := true;
            end else PrintHeaders;
          end;
        end else if not SkipBlock then
OutVectorLine(Line,InFile);
```

```
      end;  {while}
    end;  {for}
    Blk := '#BTM';
    MakeControl;
    LastLayer := MergeLayer;
    MergeLayer := 99999;
    for InFile := 1 to NumPartFiles do
      if not eof(InputFiles[InFile]) then
        if CurrentZ[InFile] < MergeLayer then MergeLayer
:= CurrentZ[InFile];
  until MergeLayer = 99999;
  if BinaryOutput then WriteVectorCount;
end;

procedure Update(FileName: String);
var
  i: Integer;
  St: String;
begin
  assign(UpdateFile,FileName);
  reset(UpdateFile);
  Seek(UpdateFile,UpdateIndex);
  St := WordStr(StartLayer);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 16);
  St := WordStr(EndLayer);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 16 + 17);
  St := WordStr(Layers);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 60);
  St := WordStr(MinX);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
```

```
  Seek(UpdateFile,UpdateIndex + 60 + 16);
  St := WordStr(MaxX);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 60 + 43);
  St := WordStr(MinY);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  Seek(UpdateFile,UpdateIndex + 60 + 59);
  St := WordStr(MaxY);
  St := St + spc(5-length(St));
  for i:=1 to 5 do write(UpdateFile,St[i]);
  close(UpdateFile);
end;

{-----------------------------------------------------}

procedure MergeFiles;
begin
{ determine output file prefix }
  if OutFileName = '' then begin
    write('Output File Name Prefix [',FirstFileName,']:
');
    readln(OutFileName);
    if OutFileName = '' then OutFileName :=
FirstFileName;
    OutFileName := upper(OutFileName);
  end;

{ open up output files, existing files are overwritten }
  LayerFileName  := OutFileName + '.L';
  RangeFileName  := OutFileName + '.R';
  VectorFileName := OutFileName + '.V';
  assign(LayerFile,LayerFileName);
  rewrite(LayerFile);
  writeln(LayerFile,'!MERGE: -FILES ',MergeLine);
```

```
  writeln(LayerFIle,'!MERGE: -STARTLAYER XXXXX -ENDLAYER
XXXXX -NUMLAYERS XXXXX');
  writeln(LayerFile,'!MERGE: -MIN X      XXXXX -MAX X
XXXXX');
  writeln(LayerFile,'!MERGE: -MIN Y      XXXXX -MAX Y
XXXXX');
  UpdateIndex := length(MergeLine) + 17 + 20;   { points
to StartLayer value }
  assign(RangeFile,RangeFileName);
  rewrite(RangeFile);
  writeln(RangeFile,'!MERGE: -FILES ',MergeLine);
  writeln(RangeFile,'!MERGE: -STARTLAYER XXXXX -ENDLAYER
XXXXX -NUMLAYERS XXXXX');
  writeln(RangeFile,'!MERGE: -MIN X      XXXXX -MAX X
XXXXX');
  writeln(RangeFile,'!MERGE: -MIN Y      XXXXX -MAX Y
XXXXX');

  Layers := 0;
  MinX:=65535;
  MinY:=65535;
  MaxX:=0;
  MaxY:=0;
  FindFirstLayers;
  GetParameters;
  if BinaryOutput then begin
    assign(VectorBinaryFile,VectorFileName);
    rewrite(VectorBinaryFile,1);
    rc:=0;
    BlockWrite(VectorBinaryFile,rc,1);
  end else begin
    assign(VectorTextFile,VectorFileName);
    rewrite(VectorTextFile);
  end;

  window(StatusBoxLeft+1,StatusBoxTop+1,StatusBoxRight-
1,StatusBoxBottom-1);
```

```
  ClrScr;
  window(1,1,80,25);
  SetUpStatusBox;
  gotoxy(StatusBoxLeft+2,StatusBoxTop+1);
  write('Different Z-Spacing Ranges:');
  window(StatusBoxLeft+2,StatusBoxTop+3,StatusBoxRight-
2,StatusBoxBottom-3);
  ReadSlices;

  if BinaryOutput then close(VectorBinaryFile)
  else close(VectorTextFile);
  OutputRange;
  close(LayerFile);
  close(RangeFile);
  Update(LayerFileName);
  Update(RangeFileName);

{} if quikfixcnt <> 1 then begin
     gotoxy(savedx,savedy);
     write(RangeLayerCount+1:5);  { RangeLayerCount of
last layer }
  end;

  window(1,1,80,25);
  gotoxy(StatusBoxLeft+2,StatusBoxBottom-1);
  write(RangeCount,' range');
  if RangeCount>1 then begin
     write('s');
  end;
  write(', ',Layers,' layer');
  if Layers>1 then write('s');
  write(' processed.');
  gotoxy(1,24);
end;

{-----------------------------------------------------}
```

```
procedure CheckForOptions;
var
  i,oi,j,k: Integer;
  FileName: String;
begin
  FirstFileName := '';
  CmdLine := upper(CmdLine);
  RemoveDoubleSpaces(CmdLine);
  i := pos ('/OUTFILE',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,8);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
inc(i);
    j := FindStr(i,CmdLine,' /');
    OutFileName := copy(CmdLine,i,j-i);
    delete(CmdLine,oi,j-oi);
  end;
  i := pos('/RES',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,4);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
inc(i);
    j := FindStr(i,CmdLine,' /');
    {$R-}
    val(copy(CmdLine,i,j-i),Resolution,code);
    {$IFDEF RNGON}
      {$R+}
    {$ENDIF}
    delete(CmdLine,oi,j-oi);
  end;

  i := pos('/ZSPC',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,5);
```

```
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
inc(i);
    j := FindStr(i,CmdLine,' /');
    {$R-}
    val(copy(CmdLine,i,j-i),ZSpacing,code);
    {$IFDEF RNGON}
      {$R+}
    {$ENDIF}
    delete(CmdLine,oi,j-oi);
  end;

  i := pos('/ASCII',CmdLine);
  BinaryOutput := True;
  if i>0 then begin
    BinaryOutput := False;
    delete(CmdLine,i,6);
  end;

  i := pos('/THICK',CmdLine);
  if i>0 then begin
    oi := i;
    inc(i,6);
    while (i<length(CmdLine)) and (CmdLine[i]=' ') do
inc(i);
    j := FindStr(i,CmdLine,' /');
    val(copy(CmdLine,i,j-i),Thickness,code);
    delete(CmdLine,oi,j-oi);
  end;
  RemoveDoubleSpaces(CmdLine);
  MergeLine := CmdLine;

{ pick out slice file names and open them up }
  while CmdLine<>'' do begin
    i := pos(' ',CmdLine);
    if i=0 then begin
      FileName := CmdLine;
      CmdLine := '';
```

```
    end else begin
      FileName := copy(CmdLine,1,i-1);
      CmdLine  := copy(CmdLine,i+1,255);
    end;
    inc(NumPartFiles);
  {}
    i := pos('/X',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
  do inc(i);
      j := FindStr(i,FileName,' /');
     {
  val(copy(FileName,i,j-i),XOffset[NumPartFiles],code); }
      if i<j then OffsetX[NumPartFiles] :=
  LongVal(copy(FileName,i,j-i),k);
      delete(FileName,oi,j-oi);
    end;

    i := pos('/Y',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
  do inc(i);
      j := FindStr(i,FileName,' /');
      if i<j then OffsetY[NumPartFiles] := LongVal(copy
  (FileName,i,j-i),k);
      delete(FileName,oi,j-oi);
    end;

    i := pos('/Z',FileName);
    if i=0 then i := pos('/L',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
```

```
      while (i<length(FileName)) and (FileName[i]=' ')
do inc(i);
      j := FindStr(i,FileName,' /');
      val(copy(FileName,i,j-i),ZOffset[NumPartFiles],
code);
      delete(FileName,oi,j-oi);
    end;
    i := pos('/N',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
do inc(i);
      j := FindStr(i,FileName,'/');
      NoBlocks[NumPartFiles] :=
','+copy(FileName,i,j-i)+',';
      delete(FileName,oi,j-oi);
    end;

{**** OFFSET Option. Offsets given file by X,Y on output
    OffsetX[NumPartFiles] := 0;
    OffsetY[NumPartFiles] := 0;
    i := pos('/OFFSET',FileName);
    if i>0 then begin
      oi := i;
      inc(i,7);
      while (i<length(FileName)) and (FileName[i]=' ')
do inc(i);
      j := FindStr(i,FileName,'/');
      OffsetX[NumPartFiles] :=
LongVal(copy(FileName,i,j-i),k);
      if k<>0 then begin
        i := i + k;
        if i<j then OffsetY[NumPartFiles] :=
LongVal(copy(FileName,i,j-i),k);
      end;
      delete(FileName,oi,j-oi);
```

```
    end;  *****}

    i := pos('/O',FileName);
    if i>0 then begin
      oi := i;
      inc(i,2);
      while (i<length(FileName)) and (FileName[i]=' ')
do inc(i);
      j := FindStr(i,FileName,'/');
      OnlyBlocks[NumPartFiles] :=
','+copy(FileName,i,j-i)+',';
      delete(FileName,oi,j-oi);
    end;
    if FirstFileName = '' then FirstFileName :=
FileName;
    i := pos('.',FileName);
    if i=0 then FileName := FileName + '.SLI';
    assign(InputFiles[NumPartFiles],FileName);
    {$I-}
    reset(InputFiles[NumPartFiles]);
    {$I+}
    if IOresult<>0 then begin
      writeln('Error: Can''t open file ',FileName);
      halt(1);
    end;
  end; {while}
  if NumPartFiles = 0 then halt(0);
end;

procedure ShowFiles(FileSpec: String);
var
  FileCount: Integer;
  DirInfo: SearchRec;
begin
  FileCount := 0;
  FindFirst(FileSpec,AnyFile,DirInfo);
  if DosError<>0 then writeln('- none -')
```

```
  else begin
    while DosError=0 do begin
      if FileCount>0 then write(spc(11-((whereX-1) mod
10+1)));
      inc(FileCount);
      write(copy(DirInfo.Name,1,pos('.',DirInfo.Name)
-1));
      FindNext(DirInfo);
    end;
    writeln;
  end;
end;

procedure ProcessCommandLine;
begin
  ReadCommandLine(CmdLine);
  if CmdLine = '' then begin
    writeln('Directory Listing of Sliced Files:');
    writeln;
    ShowFiles('*.SLI');
    writeln;
    write('Slice File Names: ');
    readln(CmdLine);
    writeln;
  end;
  NumPartFiles := 0;
  CheckForOptions;
end;

procedure ReadBlockIdFile;
var
  BlockIdFile: text;
  BlockIdLine: string;
  index,Group: Integer;
  Block: String;
begin
  BlockMnemonicsCount := 0;
```

```
  {$I-}
  assign(BlockIdFile,BlockIdFileName);
  reset(BlockIdFile);
  if IOresult<>0 then begin
    assign(BlockIdFile,HomeDirectory+'\'+BlockIdFile
Name);
    reset(BlockIdFile);
    if IOresult<>0 then begin
      writeln('Can''t find Block Id file.');
      halt(2);
    end;
  end;
  {$I+}
  while not eof(BlockIdFile) do begin
    readln(BlockIdFile,BlockIdLine);
    if (length(BlockIdLine)>0) and (BlockIdLine[1]<>'!')
then begin
      index := 1;
      UnitValShortInt(Group,BlockIdLine,index);
      UnitStr(Block,BlockIdLine,index);
      if Group=0 then begin
        inc(BlockMnemonicsCount);
        BlockMnemonics[BlockMnemonicsCount] :=
upper(Block);
      end;
    end;
  end;
  close(BlockIdFile);
end;

procedure UserWait;
var
   tempkey : char;
begin
   writeln;
   writeln('   ... press any key to continue');
   repeat delay(100) until keypressed;
```

```
   tempkey := upcase(Readkey);
end;

procedure CreatePartPRMfile;
const
    anyfile         = $3F;
    TempPRMfilename= 'TEMP.PRM';
var
    File1           : Text;
    InFile,OutFile  : Text;
    Comment, Line   : String;
    ShortStr        : String;
    PartPRMfilename: String;
    i,jj,kk,index   : integer;
    PartScale       : real;
    DirInfo         : searchRec;
    XwidthVat,YwidthVat,XvatOffset,YvatOffset : longint;
    MinXvat,MinYvat,MaxXvat,MaxYvat           : longint;

begin
   { Open part.PRM, if not found create it from
Build.PRM.
     Use MinMax from part.R, compute, and store the
proper XY and viewport offsets into part.PRM. }
   FindFirst(OutFileName,anyfile,DirInfo);
   if DosError = 0 then PartPRMfilename := OutFileName
   else begin
      FindFirst('Build.PRM',anyfile,DirInfo);
      if DosError = 0 then PartPRMfilename:='Build'
      else begin
         FindFirst('\3DSYS\Build.PRM',anyfile,DirInfo);
         if DosError =0 then
PartPRMfilename:='\3DSYS\Build'
         else begin
            window(1,1,80,25);
            ClrScr;
            gotoxy(1,21);
```

```
            writeln;
            writeln('   ERROR: Build.PRM not found.  ');
            UserWait;
            Halt(1);
          end;
        end;
    end;
    PartPRMfilename := PartPRMfilename + '.PRM';
    { $ I - COMMENT...
    assign(file1,PartPRMfilename);
    reset(file1);
    i := 0;
    repeat
       readln(file1,shortStr);
       if (shortStr[1] <> '!') and (shortStr[1] <> '"')
and
          (shortStr[1] <> ' ') and (length(shortStr) <>
0)
       then begin
          i := i + 1;
          if i = 3 then XYonlyScale := realVal(shortStr,
jj);
       end;
    until i = 3;
    close(file1);
    $ I + ENDCOMMENT}
    PartScale := 3556 / Resolution;{3556 mirror bits /
# slice units per inch}
    XwidthVat := trunc(PartScale*(maxX-minX));{ slice
units to vat units }
    YwidthVat := trunc(PartScale*(maxY-minY));{ vat units
of box }
    XvatOffset := 32767 - (XwidthVat div 2) -
trunc(PartScale * minX);
    YvatOffset := 32767 - (YwidthVat div 2) -
trunc(PartScale * minY);
```

```
   if (XvatOffset < 1000) or (YvatOffset < 1000) then
exit;
   MinXvat := 32767 - trunc((XwidthVat div 2) * 1.09);
   MinYvat := 32767 - trunc((YwidthVat div 2) * 1.09);
   MaxXvat := 65535 - MinXvat;
   MaxYvat := 65535 - MinYvat;
{***   writeln('PartScale=',PartScale:5:5);
   writeln('Resolution=',Resolution:6:5);
   writeln('minX=',minx,' minY=',minY,' maxX=',maxX,'
maxY=',maxY);
   writeln('XwidthVat=',XwidthVat,'
YwidthVat=',YwidthVat);
   writeln('XvatOffset=',XvatOffset,'
YvatOffset=',YvatOffset);
   writeln('MinXvat=',MinXvat,' MinYvat=',MinYvat);
   writeln('MaxXvat=',MaxXvat,' MaxYvat=',MaxYvat);
   userwait;  ***}
   if (MinXvat < 0) or (MinYvat < 0) or
      (MaxXvat > 65535) or (MaxYvat > 65535) then exit;
   if MinXvat > MinYvat then MinXvat := MinYvat   { we
want a box viewport }
                        else MinYvat := MinXvat;
   if MaxXvat < MaxYvat then MaxXvat := MaxYvat
                        else MaxYvat := MaxXvat;
{-------------------------------------------------}
    {$I-}
    assign(InFile,PartPRMfilename);
    reset(InFile);
    assign(OutFile,TempPRMFileName);
    rewrite(OutFile);
    repeat
      readln(InFile,line);
      if (Length(line)=0) or (line[1]='!') or
(line[1]='"')
      then writeln(OutFile,line);
    until (Length(line)>0) and (line[1]<>'!') and
(line[1]<>'"');
```

```
    i := 0;
    repeat
      i := i + 1;
      if (i=3) or (i=4) or (i=5) or (i=7) or (i=8) then
begin
        repeat
          if (Length(line)=0) or (line[1]='!') then
writeln(OutFile,line);
        until (Length(line)>0) and (line[1]<>'!');
        Index := 1;
        while ((line[index]>='0') and
(line[index]<='9')) or
              (line[index]='.') or (line[index]=',') or

              (line[index]=' ') do Inc(Index);
{ first char of comment }
        Comment := Copy(line,Index,255);
        {****}
        if i = 3 then begin  { XY only scale factor }
           str(PartScale:2:4,ShortStr);
           for jj:= 1 to length(ShortStr) do
line[jj]:=ShortStr[jj];
        end;
        if i = 4 then begin
           str(XvatOffset,ShortStr);
           for jj := 1 to length(ShortStr) do
line[jj]:=ShortStr[jj];
        end;
        if i = 5 then begin
           str(YvatOffset,ShortStr);
           for jj := 1 to length(ShortStr) do
line[jj]:=ShortStr[jj];
        end;
        if i = 7 then begin
           str(MinXvat,ShortStr);
```

```
                for jj := 1 to length(ShortStr) do
        line[jj]:=ShortStr[jj];
                kk := jj+2;
                line[kk-1]:=',';
                str(MinYvat,ShortStr);
                for jj := kk to (kk-1+length(ShortStr)) do
                        line[jj]:=ShortStr[jj-kk+1];
            end;
            if i = 8 then begin
                str(MaxXvat,ShortStr);
                for jj := 1 to length(ShortStr) do
        line[jj]:=ShortStr[jj];
                kk := jj+2;
                line[kk-1]:=',';
                str(MaxYvat,ShortStr);
                for jj := kk to (kk-1+length(ShortStr)) do
                        line[jj]:=ShortStr[jj-kk+1];
            end;
            {****}
            for kk := (jj+1) to (index-1) do line[kk]:=' ';
        { clear field }
            for kk := index to length(Comment) do
                    line[kk] := Comment[kk-index+1];
            writeln(OutFile,line);   { output reassembled
        line}
          end else writeln(OutFile,line);
          readln(InFile,line);
        until i = 8;
        while Not Eof(InFile) do begin
          writeln(OutFile,line);
          readln(InFile,line);
        end;
        writeln(OutFile,line);
        Close(InFile);
        Close(OutFile);

        PartPRMfilename := OutFileName + '.PRM';
```

```
    {$M 16384,0,0 }
    Exec('\COMMAND.COM','/C COPY ' + TempPRMFileName + '
' + PartPRMfilename);
    Exec('\COMMAND.COM','/C ERASE ' + TempPRMFileName);
    {$I+}

end; { procedure CreatePartPRMfile }



{ --------------- MAIN PROGRAM ----------------- }


begin
  CheckBreak := true;
  FillChar(FirstVar,ofs(LastVar)-Ofs(FirstVar),#0);
{ clear variables }
  ExitSave := ExitProc;
  ExitProc := @ExitMerge;
  ClrScr;
  SingleBar := many(#196,26);
  centerln('MERGE, Sliced Files Merger');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');
  centerln(SingleBar);

  writeln;
  writeln;
  OutFileName := '';
  ReadBlockIdFile;
  ProcessCommandLine;
  MergeFiles;
  CreatePartPRMfile;
  QuietSystemError('Success.');
end.
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================

   TESTPART.PAS
   SLA-1 Test Part Utility Software

---------------------------------------------------------

   Recent History:
    1/11/88  Ver 2.50  first release
    1/25/88  Ver 2.60  version update only
    3/09/88  Ver 2.80  added copyright notice
    4/11/88  Ver 3.00  production release
    5/17/88  Ver 3.10  upgraded to turbo pascal version
4.0
    7/18/88  Ver 3.20  update VersionId
---------------------------------------------------------}

uses CRT, DOS;

{$I-,R-,S+,I-,F-,V+,B- }
{$M 4096,0,0}

const
  VersionId  = '3.20';

type
  str80 = string[80];

var
  key, Mchar          : char;
  z                   : integer;
  file2, file3        : text;
  SPstring            : Str80;
  i,j, valcode        : integer;
  xside,yside,X,Y     : real;
```

```
  SP, SPlargest, cx, cy, scale, yinc : real;

procedure UserWait;
begin
   writeln;
   writeln('   USERWAIT : ... press key to continue ');
   repeat delay(100) until Keypressed;
   key := readkey;
   writeln('************************');
   writeln('************************');
   delay(2000);
end;

procedure ShowTop;
begin
  window(1,1,80,25);
  ClrScr;

writeln('-----------------------------------------');
  writeln;
  writeln('TESTPART, Test Part Utility');
  writeln;
  writeln('Version ',VersionID);
  writeln('Copyright (C) 1988 by 3D Systems, Inc. ');
  writeln('3D Systems Laser Stereolithography System ');
  writeln;
  writeln('-----------------------------------------');

  writeln;
end;

procedure MainScreen;
begin
  ShowTop;
  writeln;
  writeln('          1. BanjoTop, Determine Working
Curves');
```

```
  writeln;
  writeln('          X. Exit Program');
  writeln;
  repeat key:=upcase(ReadKey) until (key='X') or
(key='1');
  if key='X' then Halt;
end;

function RealVal(line: string;var code1: Integer):Real;
var
   code2  : integer;
   num    : real;
begin
   if line[1]='.' then line := '0' + line;
   if (line[1]='-') and (line[2]='.') then begin
      line := '-0' + line;
      for i:= 4 to length(line) do line[i-1]:=line[i];
      line[length(line)] := ' ';
   end;
   val(line,num,code1);
   if code1 <> 0 then
val(copy(line,1,code1-1),num,code1); { was code2 }
   RealVal := num;
end;


procedure SetUp;
begin
   cx := 32000;                  {  approx center of vat
}
   cy := 32000;
   scale := 140;                 {  bits/mm           }
   xside := 25;                  {  1 x 1/2 inch      }
   xside := xside * scale;
   yside := xside / 2;
   sp := 500;                    {  base step period   }
   yinc := yside  / 4;           {  overlap            }
```

```
end;

procedure ShowBJTTop;
begin
   writeln;
   writeln('          BanjoTop : Determine Working
Curves');
   writeln;
end;

procedure ShowSPs;
begin
   SP := SPlargest;
   writeln;
   writeln('  The following step periods will be
generated:');
   writeln;
   for i := 1 to 5 do begin
      j := round(SP);
      writeln('   step period ',i,' =',j);
      SP := SP / 2;
   end;
   writeln;
end;

procedure mainBJTheader;
begin
   ShowTop;
   ShowBJTTop;
   ShowSPs;
end;

procedure StartBanjoTop;
begin
   ShowTop;
   ShowBJTTop;
   repeat
```

```
      writeln('   Input largest step period value (160
minimum) of the 5 to be drawn:');
      readln(SPstring);
      if (SPstring[1] = 'X') or (SPstring[1] = 'Q') then
Halt;
      SPlargest := RealVal(SPstring,valcode);
      if (SPlargest / 16) < 10 then begin
         writeln;
         writeln('   Error: step period (SP) value input
is less than 160.');
         writeln;
      end;
   until SPlargest / 16 >= 10;
  ShowSPs;
   writeln('   OK ?  ');
   repeat key:= upcase(ReadKey) until (key='Y') or
(key='N');
   if key='N' then exit;
   mainBJTheader;
   writeln;
   writeln;
   writeln('               please wait...             ');
   exec('c:\command.com','/c CLIFF ''HR/W/T');  { Hard
Reset }
   delay(2000);
end;

procedure OpenFile;
begin
   assign(file2,'laser.dat');
   reset(file2);
   rewrite(file2);
   if (IOResult<>0) then begin
      writeln('   Error opening file LASER.DAT, program
terminating.');
      Halt;
   end;
```

```
end;

procedure writeXYtoFile;
begin
   writeln(file2,MChar,'X ',cx+x:5:1);
   writeln(file2,MChar,'Y ',cy+y:5:1);
end;

procedure WriteBase;
begin
   MChar := 'J';
   X := xside / 2;
   Y := yside / 2 + yinc;
   writeXYtoFile;
   MChar := 'N';
   X :=  xside / 2;
   Y := -yside / 2;
   writeXYtoFile;
   X := -xside / 2;
   Y := -yside / 2;
   writeXYtoFile;
   X := -xside / 2;
   Y :=  yside / 2;
   writeXYtoFile;
   X :=  xside / 2 + yinc;
   Y :=  yside / 2;
   writeXYtoFile;
end;

procedure WriteStrings;
   var   Xinteger : LongInt;
begin
   SP := SPlargest * 2;
   X  := -3 * xside / 8;
   for i := 1 to 5 do begin
      SP := SP / 2;
      j := round(SP);
```

```
      writeln(file2,'SP ',j);
      MChar:='J';
      Y := -yside / 2 - yinc;
      WriteXYtoFile;
      MChar:='N';
      Y :=  yside / 2 + yinc;
      writeXYtoFile;
      X := round( X + (6 * xside / (12*4)));  { this was
done in a basic step loop }
   end;
end;

procedure WriteParameters;
begin
   writeln(file2,'MD BL +');
   writeln(file2,'CL');
   writeln(file2,'SS2');
   writeln(file2,'SP 500');  { this is from the SP in
procedure SetUp }
   writeln(file2,'JD0');
   writeln(file2,'BX32000');
   writeln(file2,'BY62000');
end;

procedure MoveMirrors;
begin
   writeln(file2,'EX');
   close(file2);
   exec('c:\command.com','/c CLIFF LASER.DAT/W/T');
   delay(500);
   if DOSerror = 0 then writeln('   Return Code =
',DOSexitcode);
   delay(1000);
   assign(file3,'Cliff.ERR');
   reset(file3);
   if (IOResult = 0) then begin
```

```
      writeln('   Error(s) have been detected and
written to CLIFF.ERR, ');
      writeln('   CLIFF process aborted.        ');
      close(file3);
   end;
end;

begin
   MainScreen;
   Setup;
   repeat StartBanjoTop until key='Y';
   MainBJTheader;
   OpenFile;
   WriteParameters;
   WriteBase;
   WriteStrings;
   gotoxy(18,27);
   writeln('making base and strings...   ');
   MoveMirrors;
   gotoxy(18,27);
   writeln('testpart BanjoTop complete.   ');
   delay(2000);
end.^Z
```

```
Program SLAGRAPH;
(
                        3D Systems SLA1 Graphics Software
                        ---------------------------------
                        Copyright (C) 1988 3D Systems Inc.


  Change Log:
    2/15/88  Test260  Initial Pascal version
    2/18/88    2.50   Adjust 3D graph scaleing
                      Change layer select of graph for
PgUp and PgDn
                      Correct ASC Error handling
    2/19/88    2.60   Load Graphic drivers into EXE
               2.61   Change command lines to () format
                      Accept either ENDSOLID or END
SOLID in ASCII files
                      Remove ref. to ARIES
    2/26/88    2.62   Correct FILE to LAYER in slice
file menu
    3/02/88    2.70   Change name from SLAGRAPH to GRAPH

                      Correct spelling (Graphis to
Graphics)
                      Read last line of text file even
if no CR
                      Rotation for triangles -360 to 360

                      RHR for axis rotation
                      Add Axis on triangle screen
    4/06/88    3.00   Production Release, change menus
and support axis switch
    4/29/88    3.10   Add pan, zoom, delay, Xtended
viewing mode, Surface mode,
                      Normal check mode, and more.
```

```
    6/22/88     3.11  Change pan from pixel input to -10
to +10 input
    6/27/88     3.12  Support EGA, VGA, and Hercules
displays, add rectangles
    7/18/88     3.20  Update Version
    7/25/88     3.21  In surface mode, ignore all facets
whose normals are <= 0.
    8/12/88     3.22  Add VGA color support
    8/17/88     3.23  Change main menu #3, and center
all options
    8/26/88     3.24  Allow '.5' string by using
function RealVal, correct
                      begin/end facet setting, and
correct pausing if a
                      range is selected.
    9/09/88     3.25  fix totaltriangles in select range
pause/quit, and rot.
    9/14/88     3.26  make number of facets (max) =
total triangles, not tt-1
}
{$R-,S-,I+,D-,T-,F-,V-,B-,N-,L- }
{$M 16384,0,655360 }

  uses
    Crt,
    Dos,
    Graph,
    Drivers;

  const

    VERSION : String = '3.26';
    MAX_WORD : Word = 65535;
    MAX_REAL : Real = 1.7E37;
    MIN_REAL : Real = -1.7E37;
    ZERO_DEF = 1E-5;
    ESC      : Char = #27;
```

```
    CR        : Char = #13;
    LF        : Char = #10;
    EOF_CHAR : Char = #26;
    BUF_SIZE = 8191;
    Spaces : String =        '                              ';
    MAX_PTR_ARRAY = 16382;

    { Slice Constants }
    MAX_LAYERS     = 16382;
    LAYER_SIZE     = 16;
    MAX_SECTIONS   = 5;
    SliceBlocks : Array [0..4] of String[3] =
('L','FU','FD','NFU','NFD');
    L_SECTIONS     = $0007;
    L_SECTION      = $8000;
    LB_SECTION     = $0001;
    LH_SECTION     = $0002;
    FU_SECTIONS    = $0038;
    FD_SECTIONS    = $01C0;
    NFU_SECTIONS   = $0E00;
    NFD_SECTIONS   = $7000;
    F_SECTIONS     = $01F8;
    NF_SECTIONS    = $7E00;

    { Triangle Constants }
    MAX_TRIANGLES  = 10921;
    TRIANGLE_SIZE  = 10;
    ASC_FILE       = 1;       { File Type Constants }
    BINARY_FILE    = 2;
    ARIES_FILE     = 3;
    COOR_CHAR : Array [1..3] of Char = ('X','Y','Z');

  type
    DynamicSLWordArray     = Array [0..16382] of Word;
    DynamicSLLongIntArray  = Array [0..16382] of
LongInt;
```

```
    DynamicTRILongIntArray = Array [0..10921] of
LongInt;
    DynamicTRIRealArray    = Array [0..6552] of Real;
    PointerStruct          = Array [0..255] of Byte;
    PointerType            = ^PointerStruct;

  var
    ExitSave                              : Pointer;
    HeapTop                               : Pointer;
    InBuffer                              : Array
[0..BUF_SIZE] of Char;
    InBufStr                              : String;
    InBufEnd                              : Integer;
    InBufPtr                              : Integer;
    CurrFilePos                           : LongInt;
    StartLinePos                          : LongInt;
    LinePosValid                          : Boolean;
    CurrLinePos                           : LongInt;
    EndOfFile                             : Boolean;
    FileError, SSorted, WSorted           : Boolean;
    ErrLine                               : String;
    ch, DisplayMode                       : Char;
    rc                                    : Integer;
    iw,jw                                 : Word;
    TempInt                               : Integer;
    TempByte                              : Byte;
    TempWord                              : Word;
    TempBoolean                           : Boolean;
    TempReal                              : Real;
    TempLong                              : LongInt;
    TempStr,TempStr1                      : String;
    TempChar,DisplayKey                   : Char;
    InputLine                             : String;
    MonoDisplay, StopDisplayLoop          : Boolean;
    GraphDriver, GraphMode                : Integer;
    SizeX,SizeY                           : Integer;
    AspectRatio                           : Real;
```

```
    GraphScale                          : Real;
    GraphCenterX,GraphCenterY           : Word;
    GrOffsetX,GrOffsetY,GrOffsetZ       : Real;
    GF2D                                : Array
[1..3,1..2] of Integer;
    DX,DY,DisplayDelay,HoriOffset,VertOffset      :
integer;
    InputStr                            : String;

InputInt,ValCode,BeginFacet,EndFacet,NormLength,Xpixels,
Ypixels : integer;
    zoom,barN                           : real;
    Vector3,Vector1,SliceNormal,DeltaNormal       :
array[1..3] of real;
    NormalsAreOK                         : boolean;
{   gftest                              : text; }
    regs                                : registers;
    { Triangle FIle Global Vars }
    TriangleFile                        : File;
    TriangleFilename                    : String;
{ Filename of Triangle File  }
    TriangleFiletype                    : Byte;
{ File Type of Input File }
    TrianglePtr                         :
^DynamicTRILongIntArray; { Pointers to Triangle File }
    TriangleAxisValue                   :
^DynamicTRIRealArray;    { Z Values of each triangle }
    TriangleNormal                      : Array [1..3] of
Real;
    TriangleVertex                      : Array
[1..3,1..3] of Real;
    SavePastVertex                      : Array
[1..3,1..3] of Real;
    TriangleAttrib                      : Word;
    TriangleTotal                       : LongInt;
    TriangleFileMin                     : Array [1..3] of
Real;
```

```
    TriangleFileMax                    : Array [1..3] of
Real;
    TriangleSelected                   : LongInt;
    TriangleSelectStart                : Real;
    TriangleSelectEnd                  : Real;
    TriangleSelectMin                  : Array [1..3] of
Real;
    TriangleSelectMax                  : Array [1..3] of
Real;
    TriangleWindowingFile              : Boolean;
    TriangleAngle                      : Array [1..3] of
Integer;
    TriangleAxis                       : Byte;
    Gr3DMat                            : Array
[0..2,0..2] of Real;
    Total_Triangles                    : LongInt;

    { Slice File Global Vars }
    SliceFile                          : File;
    SliceFilename                      : String;
{ Filename of Slice File  }
    SliceFileXMin,SliceFileXMax        : Word;
{ File Max and Min values }
    SliceFileYMin,SliceFileYMax        : Word;
    SliceLastLayer                     : Integer;
{ Pointer to last layer   }
    SliceLayerValue                    :
^DynamicSLWordArray;    { Layer Values }
    SliceLayerPtr                      :
^DynamicSLLongIntArray; { Pointer to Layer in file }
    SliceLayerBlocks                   :
^DynamicSLWordArray;    { Layer Blocks }
    SliceLayerXMin,SliceLayerXMax      :
^DynamicSLWordArray;    { Layer Max and Min values }
    SliceLayerYMin,SliceLayerYMax      :
^DynamicSLWordArray;
```

```
    SliceLayer                          : Word;
  { Current Layer }
    SliceWindowingFile                  : Boolean;
    SliceOutputBlocks                   : Word;
    Total_Layers                        : LongInt;

{$I view.inc}

{-------------------------------------------------------
    |     Start of Main Triangle Routines          |

-------------------------------------------------------}

function ReadTriangle : Boolean;
  { Read in one triangles data from input files current
location }
  var
    ParsingLine : String;
    i,j     : Word;
  begin
    if TriangleFiletype=ASC_FILE then begin
      TriangleAttrib:=0;
      ReadLine(TriangleFile); { FACET NORMAL i j k }
      TempStr:=ParseLine(InputLine);
      if Copy(TempStr,1,3)='END' then EndOfFile:=TRUE
      else begin
        if (TempStr<>'FACET') OR
(ParseLine(InputLine)<>'NORMAL') then FileError:=TRUE;
        if Not FileError then for i:= 1 to 3 do begin
          Val(ParseLine(InputLine),TriangleNormal[i],
rc);
          if rc<>0 then FileError:=TRUE;
        end;
        ReadLine(TriangleFile); { OUTER LOOP }
        if (ParseLine(InputLine)<>'OUTER') OR
(ParseLine(InputLine)<>'LOOP') then FileError:=TRUE;
        if Not FileError then for i:= 1 to 3 do begin
```

```
          ReadLine(TriangleFile); { VERTEX }
          if ParseLine(InputLine)<>'VERTEX' then
FileError:=TRUE
          else for j:= 1 to 3 do begin
            Val(ParseLine(InputLine),TriangleVertex
[i,j],rc);
            if rc<>0 then FileError:=TRUE;
          end;
        end;
      end;
    end else begin
      { A Binary File }
      for i:=0 to 2 do ConvertStoR(@InBuffer[InBufPtr
+i*4],@TriangleNormal[i+1]);
      if InBufEnd-InBufPtr>11 then for i:=0 to 2 do
        for j:=0 to 2 do ConvertStoR(@InBuffer
[InBufPtr+12+i*12+j*4],@TriangleVertex[i+1,j+1]);
    end;
    ReadTriangle:=Not (FileError or EndOfFile);
  end;

procedure GraphTriangles(var ch:Char);
  { Graph the selected triangles to screen }
  const
     XRedBox : array[1..4] of PointType = ((x:  1;
y:339),
                                           (x:  1;
y:286),
                                           (x:280;
y:286),
                                           (x:280;
y:339));
     RedBox : array[1..4] of PointType =  ((x:  1;
y:350),
                                           (x:  1;
y:338),
```

```
                                                    (x:280;
y:338),
                                                    (x:280;
y:350));
  var
    CurrentColor, savePastColor, i,j  : integer;
    index : LongInt;
    min,max : Real;
    MidPoint: array [1..2,1..3] of real;
    savenormx,savenormy: Integer;
  begin
    min:=MAX_REAL;
    max:=MIN_REAL;
    Init3D;
    if TriangleWindowingFile then
      SetupGraphicR(TriangleFileMin[1],TriangleFileMax
[1],
                    TriangleFileMin[2],TriangleFileMax
[2],
                    TriangleFileMin[3],TriangleFileMax
[3])
    else
      SetupGraphicR(TriangleSelectMin[1],TriangleSelect
Max[1],
                    TriangleSelectMin[2],TriangleSelect
Max[2],
                    TriangleSelectMin[3],TriangleSelect
Max[3]);
    GrTextXY('Triangle Filename   :
'+T. angleFilename,1,1);
    Str(TriangleSelected,TempStr);
    GrTextXY('Triangle Count      : '+TempStr,1,2);
    GrTextXY('or <esc> for main menu',43,3);
    GrTextXY('Q to quit current display-',43,2);
    GrTextXY('press any key to pause-',43,1);

    index := beginFacet;
```

```
    DisplayKey := 'F'; { Step, Forward, Run, Stop=
space, Back, Quit }
    StopDisplayLoop:=false;
    SavePastColor := 0;
    CurrentColor := 7; {lightgray}
    SelectGraphColor(CurrentColor);
    DrawAxis;

    while DisplayKey<>'Q' do begin
     SelectGraphColor(red);
     SetLineStyle(SolidLn,$00,NormWidth);
     SetFillStyle(SolidFill,red);
     FillPoly(4,RedBox);
     if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
     GrTextXY(' Drawing in Process... ',1,35);
    repeat
       if ((TriangleAxisValue^[index-1]>=TriangleSelect
Start) and
         (TriangleAxisValue^[index-1]<=TriangleSelect
End))
         or (displaymode='S') then begin
        PositionFile(TriangleFile,TrianglePtr^
[index-1]);
        TempBoolean:=ReadTriangle;
        if abs(TriangleNormal[TriangleAxis])>1-ZERO_DEF
then
          if TriangleNormal[TriangleAxis]>0 then
CurrentColor:=10
               { LightGreen represents FU }
          else CurrentColor:=9
               { LightBlue represents FD}
        else if TriangleNormal[TriangleAxis]>0 then
CurrentColor:=2
               { Green represents U}
              else CurrentColor:=1;
```

```
                { Blue represents D }
            if DisplayMode = 'X' then begin
               if (SavePastColor <> 0) then begin
                  { the triangle is first drawn in red, now
it is redrawn in
                    its proper current color for the final
display - noncurrent }
                  SelectGraphColor(SavePastColor);
                  DrawPartLine(savepastVertex[1,1],savepast
Vertex[1,2],savepastVertex[1,3],
                               savepastVertex[2,1],savepastVertex
[2,2],savepastVertex[2,3]);
                  DrawPartLine(savepastVertex[2,1],savepast
Vertex[2,2],savepastVertex[2,3],
                               savepastVertex[3,1],savepastVertex
[3,2],savepastVertex[3,3]);
                  DrawPartLine(savepastVertex[3,1],savepast
Vertex[3,2],savepastVertex[3,3],
                               savepastVertex[1,1],savepastVertex
[1,2],savepastVertex[1,3]);
                  { redraw the normal in grey instead of
yellow }
                  SelectGraphColor(LightGray);
                  DrawPartLine(MidPoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                               MidPoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
                  OutTextXY(saveNormX,saveNormY,'+');
               end;
               for i:=1 to 3 do begin
                  for j:=1 to 3 do savePastVertex[i,j]:=
TriangleVertex[i,j];
                  MidPoint[1,i]:=(TriangleVertex[1,i]+
TriangleVertex[2,i]+
                                  TriangleVertex[3,i])/3.0;
               end;
               SavePastColor:=CurrentColor;
```

```
            CurrentColor := 4; (red)
            SelectGraphColor(CurrentColor);
            Vertex3DZoom(MidPoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                         saveNormX,saveNormY);
            saveNormX:=saveNormX-3;
            saveNormY:=saveNormY-3;
            SelectGraphColor(White);
            OutTextXY(saveNormX,saveNormY,'+');
            SelectGraphColor(14); { yellow }
            for i:= 1 to 3 do MidPoint[2,i]:=MidPoint
[1,i]+
                  (TriangleNormal[i]*(NormLength/100)/
Zoom);
            DrawPartLine(Midpoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                         Midpoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
            SelectGraphColor(red);
            SetFillStyle(SolidFill,red);
            FillPoly(4,XRedBox);
            if MonoDisplay then SetColor(Black) else
SelectGraphColor(white);
            InputInt := index;
            Str(InputInt:6,TempStr);
            GRTextXY(' Facet  = '+TempStr,1,30);
            Str(TriangleNormal[1]:6:3,TempStr);
            Str(TriangleNormal[2]:6:3,TempStr1);
            Str(TriangleNormal[3]:6:3,InputStr);
            GRTextXY(' NORMAL = '+TempStr+'/'+TempStr1+
'/'+InputStr,1,31);
            Str(TriangleVertex[1,1]:6:2,TempStr);
            Str(TriangleVertex[1,2]:6:2,TempStr1);
            Str(TriangleVertex[1,3]:6:2,InputStr);
            GRTextXY(' XYZ PT1= '+TempStr+'/'+
TempStr1+'/'+InputStr,1,32);
            Str(TriangleVertex[2,1]:6:2,TempStr);
```

```
            Str(TriangleVertex[2,2]:6:2,TempStr1);
            Str(TriangleVertex[2,3]:6:2,InputStr);
            GRTextXY(' XYZ PT2= '+TempStr+'/'+
TempStr1+'/'+InputStr,1,33);
            Str(TriangleVertex[3,1]:6:2,TempStr);
            Str(TriangleVertex[3,2]:6:2,TempStr1);
            Str(TriangleVertex[3,3]:6:2,InputStr);
            GRTextXY(' XYZ PT3= '+TempStr+'/'+
TempStr1+'/'+InputStr,1,34);
         end; { if extended wireframe }

         if DisplayMode='N' then begin
            SelectGraphColor(red);
            SetFillStyle(SolidFill,red);
            FillPoly(4,XRedBox);
            if MonoDisplay then SetColor(Black) else
SelectGraphColor(white);
            InputInt := index;
            Str(InputInt:6,TempStr);
            GRTextXY(' Facet  = '+TempStr,1,30);
            { TriangleNormals should be unit vectors
already. }
            Str(TriangleNormal[1]:6:3,TempStr);
            Str(TriangleNormal[2]:6:3,TempStr1);
            Str(TriangleNormal[3]:6:3,InputStr);
            GRTextXY(' FILE NORM  =
'+TempStr+'/'+TempStr1+'/'+InputStr,1,31);
            for i := 1 to 3 do begin
               Vector3[i]:=TriangleVertex[3,i]-
TriangleVertex[2,i];
               Vector1[i]:=TriangleVertex[1,i]-
TriangleVertex[2,i];
            end;
            { Vector3 cross Vector1 yields correct facet
normal as per SLICE. }
            SliceNormal[1]:= Vector3[2]*Vector1[3]-
Vector3[3]*Vector1[2];
```

```
            SliceNormal[2]:= Vector3[3]*Vector1[1]-
Vector3[1]*Vector1[3];
            SliceNormal[3]:= Vector3[1]*Vector1[2]-
Vector3[2]*Vector1[1];
            barN := SQRT(SQR(SliceNormal[1])+
SQR(SliceNormal[2])
                                                +SQR(Slice
Normal[3]));
            for i := 1 to 3 do SliceNormal[i]:=
SliceNormal[i]/barN;
            Str(SliceNormal[1]:6:3,TempStr);
            Str(SliceNormal[2]:6:3,TempStr1);
            Str(SliceNormal[3]:6:3,InputStr);
            GRTextXY(' SLICE NORM = '+TempStr+'/'+
TempStr1+'/'+InputStr,1,32);
            NormalsAreOK := true;
            for i:= 1 to 3 do begin
               DeltaNormal[i]:= ABS(Triangle
Normal[i]-SliceNormal[i]);
               if (DeltaNormal[i]>0.001) then
NormalsAreOK := false;
            end;
            if NormalsAreOK then begin
               GRTextXY(' Facet file normal is
correct.',1,34);
            end else begin
               StopDisplayLoop := true;
               GRTextXY(' *** ERROR in file normal
(dashed).',1,34);
               Str(DeltaNormal[1]:6:3,TempStr);
               Str(DeltaNormal[2]:6:3,TempStr1);
               Str(DeltaNormal[3]:6:3,InputStr);
               GRTextXY(' deviation  =
'+TempStr+'/'+TempStr1+'/'+InputStr,1,33);
               for i:=1 to 3 do begin
                  MidPoint[1,i]:=(TriangleVertex[1,i]+
TriangleVertex[2,i]+
```

```
                                    TriangleVertex[3,i])/3.0;
                end;
                Vertex3DZoom(MidPoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                                saveNormX,saveNormY);
                SelectGraphColor(White);
                OutTextXY(saveNormX-3,saveNormY-3,'+');
                SelectGraphColor(14); {yellow}
                SetLineStyle(DashedLn,$00,NormWidth);
                for i:= 1 to 3 do MidPoint[2,i]:=MidPoint
[1,i]+
                      (TriangleNormal[i]*(NormLength/100)/
Zoom);
                DrawPartLine(Midpoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                                Midpoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
                SetLineStyle(SolidLn,$00,NormWidth);
                for i:= 1 to 3 do MidPoint[2,i]:=MidPoint
[1,i]+
                      (SliceNormal[i]*(NormLength/100)/
Zoom);
                DrawPartLine(Midpoint[1,1],MidPoint[1,2],
MidPoint[1,3],
                                Midpoint[2,1],MidPoint[2,2],
MidPoint[2,3]);
             end;
             SelectGraphColor(CurrentColor);
          end; { if normal check mode  }

          if (DisplayMode <> 'S') or (MonoDisplay) then
begin
             SelectGraphColor(CurrentColor);
             if DisplayMode ='S' then
SetLineStyle(SolidLn,0,NormWidth);
             DrawPartLine(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
```

```
                    TriangleVertex[2,1],TriangleVertex
[2,2],TriangleVertex[2,3]);
            DrawPartLine(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                    TriangleVertex[3,1],TriangleVertex
[3,2],TriangleVertex[3,3]);
            DrawPartLine(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                    TriangleVertex[1,1],TriangleVertex
[1,2],TriangleVertex[1,3]);
        end; { if wireframe }

        if (DisplayMode = 'S') and (TriangleAxisValue^
[index-1]<> 1.2345678) then begin
          { 1.2345678 is a flag indicating the normal
was <= zero }
          Vertex3Dzoom(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
                    GF2D[1,1],GF2D[1,2]);
          Vertex3Dzoom(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                    GF2D[2,1],GF2D[2,2]);
          Vertex3Dzoom(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                    GF2D[3,1],GF2D[3,2]);
          if MonoDisplay then begin
             if CurrentColor=10 then SetFillStyle
(InterleaveFill,1) {FU}
             else if CurrentColor=9 then SetFillStyle
(SlashFill,1) {FD}
             else if CurrentColor=2 then SetFillStyle
(CloseDotFill,1) {U}
             else SetFillStyle(LtSlashFill,1);      {D}
          end else SetFillStyle(SolidFill,CurrentColor);
          FillPoly(3,GF2D);
        end; { if graph is for a surface model }
        if DisplayKey <> 'R' then delay(DisplayDelay);
```

```
{    write(gftest,'***');}
      end;  { is triangleaxisvalue^ in the start to end
range }
{    writeln(gftest,'Process triangle with
INDEX=',index:6,
                  ' TriangleTotal:=',TriangleTotal:6);}
      Inc(Index);
    until (KeyPressed) or
           ((Index>TriangleTotal) or) (stopDisplayLoop)
or (index>=endfacet);
    if KeyPressed then TempChar := upcase(Readkey);
    if TempChar = ESC then exit;
    SelectGraphColor(red);
    if MonoDisplay then SetFillStyle(SolidFill,1)
    else SetFillStyle(SolidFill,red);
    FillPoly(4,RedBox);
    if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
    if Index>=EndFacet then begin
       DrawAxis;
       if MonoDisplay then SetColor(Black) else
SetColor(LightGray);
       GrTextXY(' Drawing Complete: hit any to
exit',1,35);
       repeat delay(100) until KeyPressed;
       TempChar:=upcase(ReadKey);
       DisplayKey:='Q';
    end else begin
       GrTextXY(' Drawing
Paused:Step/Forward/Run?',1,35);
       if (DisplayMode <> 'S') then begin
          repeat
             { FLASHER }
           for i := 1 to 2 do begin
             delay(200);
             if (i = 1) then SelectGraphColor(Black)
             else SelectGraphColor(CurrentColor);
```

```
            if DisplayMode ='S' then
SetLineStyle(SolidLn,0,NormWidth);
            DrawPartLine(TriangleVertex[1,1],
TriangleVertex[1,2],TriangleVertex[1,3],
                  TriangleVertex[2,1],TriangleVertex
[2,2],TriangleVertex[2,3]);
            DrawPartLine(TriangleVertex[2,1],
TriangleVertex[2,2],TriangleVertex[2,3],
                  TriangleVertex[3,1],TriangleVertex
[3,2],TriangleVertex[3,3]);
            DrawPartLine(TriangleVertex[3,1],
TriangleVertex[3,2],TriangleVertex[3,3],
                  TriangleVertex[1,1],TriangleVertex
[1,2],TriangleVertex[1,3]);
          end;
         until Keypressed;
       end;
       TempChar:= upcase(ReadKey);
       if TempChar=ESC then exit;
       case TempChar of
        {  ' ' : begin
                  repeat delay(100) until KeyPressed;
                  TempChar:=upcase(ReadKey);
                  if TempChar=ESC then exit;
                  DisplayKey:=TempChar;
                end; }
          'Q' : DisplayKey:='Q';
          'R' : DisplayKey:='R';
          'F' : DisplayKey:='F';
          'S' : DisplayKey:='S';
       end;
       if DisplayKey='S' then StopDisplayLoop:=true else
StopDisplayLoop:=false;
    end;
    end; { do while }
    ch := TempChar;
  end;
```

```
procedure PrintRealRange(Str1,Str2 : String;min,max :
Real);
  { Prints Range values on line for parameters }
  begin
    GotoXY(3,WhereY);
    TextColor(LightGray);
    write(Copy(Str1+'              ',1,12),': ');
    TextColor(White);
    if min<MAX_REAL then write(min+0.0003:16:3);
    GotoXY(42,WhereY);
    TextColor(LightGray);
    write(Copy(Str2+'              ',1,12),': ');
    TextColor(White);
    if max>MIN_REAL then write(max+0.0003:16:3);
  end;

procedure PrintTriangleMenu;
  { Print out Triangle Main Menu }
  var
    i  : Word;
  begin
    PrintTriHeader;
    GotoXY(3,5);
    TextColor(LightGray);
    write('Triangle Filename : ');
    TextColor(White);
    write(TriangleFilename);
    GotoXY(47,WhereY);
    TextColor(LightGray);
    write('Triangle Filetype : ');
    TextColor(White);
    case TriangleFiletype of
      ASC_FILE    : write('ASCII');
      BINARY_FILE : write('Binary');
      ARIES_FILE  : write('Aries');
    end;
```

```
    GotoXY(3,WhereY+1);
    TextColor(LightGray);
    write('Total Triangles   : ');
    TextColor(White);
    writeln(TriangleTotal+1);
    for i:= 1 to 3 do begin
      GotoXY(1,WhereY+1);
      PrintRealRange('File '+COOR_CHAR[i]+' Min','File
'+COOR_CHAR[i]+'
Max',TriangleFileMin[i],TriangleFileMax[i]);
    end;
    GotoXY(3,WhereY+3);
    TextColor(LightGray);
    write('Total Selected : ');
    TextColor(White);
    writeln(TriangleSelected);
    PrintRealRange('Selected Min','Selected Max',
      TriangleSelectStart,TriangleSelectEnd);
    for i:= 1 to 3 do begin
      GotoXY(1,WhereY+1);
      PrintRealRange('Sel. '+COOR_CHAR[i]+' Min','Sel.
'+COOR_CHAR[i]+' Max',
        TriangleSelectMin[i],TriangleSelectMax[i]);
    end;
    TextColor(LightGray);
    GotoXY(1,WhereY+2);
    write('   Axis Rotation ( ');
    for i:=1 to 3 do write(COOR_CHAR[i],' ');
    write(') = ');
    TextColor(White);
    for i:= 1 to 3 do write(TriangleAngle[i],' ');
    writeln;
    TextColor(LightGray);
    write('   Windowing by ');
    TextColor(White);
    if TriangleWindowingFile then writeln('File')
    else writeln('Range');
```

```
        TextColor(LightGray);
        write('   Slice Axis is ');
        TextColor(White);
        write(COOR_CHAR[TriangleAxis]);
        GotoXY(49,19);
        TextColor(LightGray);
        write('Display mode=');
        TextColor(White);
        if      DisplayMode='W' then writeln(' Wireframe  ')
        else if DisplayMode='S' then writeln(' Surface    ')
        else if DisplayMode='X' then writeln(' Extended   ')
        else if DisplayMode='N' then writeln('
    NormalCheck');
        GotoXY(21,21);
        TextColor(LightGray);
        write('Horiz Offset=');
        TextColor(White);
        writeln(HoriOffset);
        GotoXY(40,21);
        TextColor(LightGray);
        write('Vertical Offset=');
        TextColor(White);
        writeln(VertOffset);
        GotoXY(62,21);
        TextColor(LightGray);
        write('Zoom = ');
        TextColor(White);
        inputint:=round(zoom*10);
        writeln(InputInt);
        TextColor(LightGray);
        GotoXY(62,20);
        write('Delay= ');
        TextColor(White);
        writeln(DisplayDelay);
        GotoXY(1,23);
        TextColor(LightGray);
```

```
    writeln('Select: New file        Zoom   Window mode
   Select range  Delay');
    writeln('        Toggle display mode   Axis of
viewing   Rotate        Graph');
      write('        Horizontal offset     Vertical
offset   Length        Begin/End');
    CursorOff;
  end;

procedure GetTriangleInfo;
  { Read in Triangle Database }
  var
    FacetCount,index1,index2 :LongInt;
    i,j        : Word;
    CurrentColor: integer;
    realvalue1 : real;
    eyedistance : real;
  begin
    Center('<Scanning Triangle File>',25);
    { Open triangle file }
    Assign(TriangleFile,TriangleFilename);
    ResetFile(TriangleFile);
    TriangleTotal:=0;
    for i:= 1 to 3 do begin
      TriangleFileMin[i]:=MAX_REAL;
      TriangleFileMax[i]:=MIN_REAL;
      TriangleSelectMin[i]:=MAX_REAL;
      TriangleSelectMax[i]:=MIN_REAL;
    end;

    if DisplayMode = 'S' then Init3D;
    repeat
      { Skip over file header }
      FacetCount:=1; { Set in case not Binary file }
      if TriangleFiletype=ASC_FILE then begin
        ReadLine(TriangleFile);
```

```
        if ParseLine(InputLine)<>'SOLID' then
FileError:=TRUE;
      end else begin
        PositionFile(TriangleFile,CurrFilePos+80);
        if TriangleFiletype=BINARY_FILE then begin
          Move(InBuffer[InBufPtr],FacetCount,4);
          PositionFile(TriangleFile,CurrFilePos+4);
        end;
      end;

      { Read in the triangles }
      while (FacetCount>0) and
(TriangleTotal<=Total_Triangles) and
            (Not (FileError or EndOfFile)) do begin
        TrianglePtr^[TriangleTotal]:=CurrFilePos;
{ Currently pointing to first triangle }
        if ReadTriangle then begin
          if (TriangleFileType=ARIES_FILE) and
(TriangleNormal[1]=0) and
             (TriangleNormal[2]=0) and
(TriangleNormal[3]=0) then begin
            FacetCount:=0;
            PositionFile(TriangleFile,CurrFilePos+
TriangleAttrib+12);
          end else begin
            if DisplayMode<>'S' then TriangleAxisValue^
[TriangleTotal]:=MAX_REAL
            else TriangleAxisValue^[TriangleTotal]:=
Min_REAL;

          for i := 1 to 3 do begin
            if DisplayMode<>'S' then begin
              if TriangleVertex[i,TriangleAxis]<
TriangleAxisValue^[TriangleTotal]
              then TriangleAxisValue^[TriangleTotal]:=
TriangleVertex[i,TriangleAxis];
            end else begin  { surface mode }
```

```
              eyedistance := ((TriangleVertex[i,1]-
GrOffsetX) * Gr3DMat[0,1])+
                                (TriangleVertex[i,2]-
GrOffsetY)*Gr3DMat[1,1]-
                                (TriangleVertex[i,3]-
GrOffsetZ)*Gr3DMat[2,1];
              if eyedistance>TriangleAxisValue^
[TriangleTotal]
              then TriangleAxisValue^[TriangleTotal]:=
eyedistance;
            end;
            for j:=1 to 3 do begin
              if TriangleVertex[i,j]<TriangleFileMin[j]
then TriangleFileMin[j]:=TriangleVertex[i,j];
              if TriangleVertex[i,j]>TriangleFileMax[j]
then TriangleFileMax[j]:=TriangleVertex[i,j];
              if TriangleVertex[i,j]<TriangleSelectMin
[j]then TriangleSelectMin[j]:=TriangleVertex[i,j];
              if TriangleVertex[i,j]>TriangleSelectMax
[j]then TriangleSelectMax[j]:=TriangleVertex[i,j];
            end; { j }
          end;
          Inc(TriangleTotal);
          if TriangleFileType=BINARY_FILE then Dec
(FacetCount);
          { Move to next triangle }
          if TriangleFiletype=ASC_FILE then begin
            ReadLine(TriangleFile);
            if ParseLine(InputLine)<>'ENDLOOP' then
FileError:=TRUE;
            ReadLine(TriangleFIle);
            if ParseLine(InputLine)<>'ENDFACET' then
FileError:=TRUE;
          end else begin
            Move(InBuffer[InBufPtr+48],TriangleAttrib,2);
            PositionFile(TriangleFile,CurrFilePos+
TriangleAttrib+50);
```

```
          end;
           end;  { else from aries file  }
         end;    { if ReadTriangle       }
       end;      { while facetcount>0... }
    until (TriangleTotal>Total_Triangles) or FileError
or EndOfFile;
    if TriangleTotal>Total_Triangles then begin
      Str(TriangleTotal,TempStr);
      Center('*** Warning: Maximum of '+TempStr+'
triangles read in. Ignoring rest. ***',24);
      WaitKey;
    end;
    if Not FileError then begin
      TriangleSelected:=TriangleTotal;
      Dec(TriangleTotal);
      if (DisplayMode = 'S') then begin
        QuickSort(0,TriangleTotal);
        for i:=0 to TriangleTotal do
{           writeln(gftest,' index=',i:6,'eyedist=',
TriangleAxisValue^[i]:8:3);}
        SSorted := true;
      end;
      TriangleSelectStart:=TriangleFileMin
[TriangleAxis];
      TriangleSelectEnd:=TriangleFileMax[TriangleAxis];
     end;

  if Displaymode = 'S' then begin
     { Go through the pointers in their quicksorted
form.  Subsort all equal eyedistances by their triangle
normal.  After this process the subsorted list of
triangles is ready to be displayed. }

     TriangleTotal := 0;
     repeat {normalsort}
{       writeln(gftest,'TOP of NORMALSORT, reset
index1=TriTotal');}
```

```
        index1:=TriangleTotal;
        eyedistance:=TriangleAxisValue^[TriangleTotal];
{        writeln(gftest,'TriTotal=',TriangleTotal,
                        ' eyedist(index1)=', eyedistance:
8:3,
                        ' TriangleSelected=',
TriangleSelected);
        writeln(gftest,'Eyedist(TriTot+1)=',
TriangleAxisValue^[TriangleTotal+1]:8:3);}
        if (TriangleAxisValue^[TriangleTotal+1]=
eyedistance) and
            (TriangleTotal+1<=TriangleSelected)
        then begin
           repeat
              inc(TriangleTotal);
           until
(TriangleAxisValue^[TriangleTotal+1]<>eyedistance) or
               (TriangleTotal+1>TriangleSelected-1) or
FileError;{ or EndOfFile;}
{           writeln(gftest,'locate index2 normsort.
FileError:=',fileerror,' EndOfFile=',endoffile);}
           index2 := TriangleTotal;
           { Do the sort, which requires computing the
normal for each triangle and storing over where
eyedistance was, then resorting from address one to
address two. }
           for i := index1 to index2 do begin
              PositionFile(TriangleFile,TrianglePtr^[i]);
              TempBoolean:=ReadTriangle;

              { this is computing the normal from the eye
view }
              { we use negative normal to sort back to
front facing }
              TriangleAxisValue^[i]:=
              (   TriangleNormal[1]*Gr3DMat[0,1]
                + TriangleNormal[2]*Gr3DMat[1,1]
```

```
                - TriangleNormal[3]*Gr3DMat[2,1] );
              if TriangleAxisValue^[i] <= 0
              then TriangleAxisValue^[i] := 1.2345678;
              { this is a flag indicating the facet need
not be drawn }
          end;   { for index1 to index2 }
          quicksort(index1,index2);
          for i:=0 to TriangleSelected-1 do
{           writeln(gftest,' index=',i:6,' normal
=',TriangleAxisValue^[i]:8:3);}
       end;
       inc(TriangleTotal);
     until (TriangleTotal>TriangleSelected-1) or
FileError;{ or EndOfFile; }
{     writeln(gftest,'END of normal sort.
FileError:=',fileerror,' EndOfFile=',endoffile);}
     dec(TriangleTotal);
  end; {normal sort for display mode of surface }
  GrOffsetX:=TriangleFileMin[1]+((TriangleFileMax[1]-
TriangleFileMin[1])/2);
  GrOffsetY:=TriangleFileMin[2]+((TriangleFileMax[2]-
TriangleFileMin[2])/2);
  GrOffsetZ:=TriangleFileMin[3]+((TriangleFileMax[3]-
TriangleFileMin[3])/2);
end;  {procedure GetTriangleInfo}

procedure DoTriangle;
  { User selected Triangle option of Main Menu }
  var
    ch        : Char;
    TempReal1 : Real;
    i,j,k     : Word;
  begin
    ch := ' ';
    FileError:=FALSE;
    repeat
      if TriangleFilename = '' then ch := 'N'
```

```
      else begin
        ResetFile(TriangleFile);
        PrintTriangleMenu;
        if ch=' ' then repeat
          ch:= UpCase(ReadKey);
        until (ch = ESC) or (Pos(ch,'NGTWSRADZHVBEL') >
0);
        TextColor(White);
      end;
      LinePosValid:=FALSE;
      case ch of
        'N' : begin
                CursorOn;
                if GetFilename(TriangleFilename,1) then
begin
                  CursorOff;
                  GotoXY(1,24);
                  write('   File "',TriangleFilename,'"
selected. File type (A)SCII or (B)inary?');
                  repeat
                    TriangleFiletype:=Pos(UpCase
(ReadKey),'ABR');
                  until TriangleFiletype>0;
                  TriangleAxis:=3; { Assume Z }
                  DX := 0;
                  DY := 0;
                  DisplayDelay:=0;
                  Zoom:=1;
                  HoriOffset:=0;
                  VertOffset:=0;
                  NormLength:=500;
                  BeginFacet:=1;
                  DisplayMode:='W';

                  GetTriangleInfo;
```

```
                    EndFacet:= TriangleSelected+1; { Wayno
3.26 }
                    WSorted:=true;
                    SSorted:=false;
                    LinePosValid:=TRUE;
                    TriangleWindowingFile:=TRUE;
                    for i:= 1 to 3 do TriangleAngle[i]:=0;
                    Xpixels := GetMaxX;
                    Ypixels := GetMaxY;
                  end;
                  ch:=' ';
                end;
        'W' : begin
                  if DisplayMode <> 'S'
                  then TriangleWindowingFile:=Not
TriangleWindowingFile;
                  ch:=' ';
                end;
        'T' : begin
                   if DisplayMode='W' then begin
                               DisplayMode:= 'X';
                                           end
                   else if DisplayMode='X' then begin
                               DisplayMode:='N';
                               BeginFacet:=1;
                               EndFacet:=
TriangleSelected+1;
                                                   end
                   else if DisplayMode='N' then begin
                                          DisplayMode:='S';
                                          BeginFacet:= 1;
                                          EndFacet:=
TriangleSelected+1;
                                          end
                   else if DisplayMode='S' then begin
                     DisplayMode:='W';
                     if WSorted = false then begin
```

```
                        close(TriangleFile);
                        GetTriangleInfo;
                        WSorted := true;
                      end;
                    end;
                    ch:=' ';
                  end;
          'S' : begin
                  if DisplayMode <> 'S' then begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Select Range Start:');
                    CursorOn;
                    readln(TempStr);
                    CursorOff;
                    ToUpper(TempStr);
                    if TempStr='RESET' then begin
                      TriangleSelectStart:=TriangleFileMin
[TriangleAxis];
                      TriangleSelectEnd:=TriangleFileMax
[TriangleAxis];
                    end else begin
                      TempReal := RealVal(TempStr,rc);
                    {   Val(TempStr,TempReal,rc);      }
                      if rc=0 then begin
                        GotoXY(1,24);
                        ClrEol;
                        write('   Select Range End:');
                        CursorOn;
                        readln(TempStr);
                        CursorOff;
                        TempReal1 := RealVal(TempStr,rc);
```

```
                    {    Val(TempStr,TempReal1,rc); }
                        if rc=0 then begin
                          if TempReal<TempReal1 then begin
                            TriangleSelectStart:=TempReal;
                            TriangleSelectEnd:=TempReal1;
                          end else begin
                            TriangleSelectStart:=TempReal1;
                            TriangleSelectEnd:=TempReal;
                          end;
                        end;
                      end;
                      for i:=1 to 3 do begin
                        TriangleSelectMin[i]:=MAX_REAL;
                        TriangleSelectMax[i]:=MIN_REAL;
                      end;
                      TriangleSelected:=0;
                      for i:=0 to TriangleTotal do begin
                        if (TriangleAxisValue^[i]>=
TriangleSelectStart) and
                           (TriangleAxisValue^[i]<=
TriangleSelectEnd) then begin
                          PositionFile(TriangleFile,
TrianglePtr^[i]);
                          TempBoolean:=ReadTriangle;
                          TempBoolean:=FALSE;
                          for j:=1 to 3 do
                            for k:= 1 to 3 do begin
                              if TriangleVertex[j,k]<
TriangleSelectMin[k] then TriangleSelectMin[k]:=
TriangleVertex[j,k];
                              if TriangleVertex[j,k]>
TriangleSelectMax[k] then TriangleSelectMax[k]:=
TriangleVertex[j,k];
                          end;
                          Inc(TriangleSelected);
                        end;
                      end;
```

```
                    end;
                    end; { displaymode <> 'S' }
                    ch:=' ';
                  end;
          'R' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter new axis rotation in ');
                    for i:=1 to 3 do write(COOR_CHAR[i],'
');
                    write(': ');
                    CursorOn;
                    readln(TempStr);
                    CursorOff;
                    GotoXY(1,25);
                    ClrEol;
                    for i:=1 to 3 do begin
                      Val(ParseLine(TempStr),TempReal,rc);
                      if rc=0 then begin
                        TempReal:=Round(TempReal);
                        if (TempReal>=-360) and
(TempReal<=360) then TriangleAngle[i]:=Round(TempReal);
                      end;
                    end;
                    SSorted := false;
                    ch:=' ';
                  end;
          'A' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    write(' Enter new axis  ( X, Y, or Z)');
```

```
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    repeat
                      TempByte:=Pos(UpCase(ReadKey),'XYZ');
                    until TempByte>0;
                    if TempByte<>TriangleAxis then begin
                      TriangleAxis:=TempByte;
                      DX:=0;
                      DY:=0;
                      if TriangleAxis=1 then DY:= -90;
{slice on X}
                      if TriangleAxis=2 then DX:= 90;
{slice on Y}
                      for i:= 1 to 3 do TriangleAngle[i]:=
0;
                      SSorted := false;
                    end;
                    ch:=' ';
                  end;
          'D' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter delay time (ms) between
triangle   ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= 0) and
(InputInt<3000)
                    then DisplayDelay:=InputInt;
```

```
                    ch:=' ';
                  end;
          'Z' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter Zoom factor (1 to 100,
where no zoom=10): ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= 1) and
(InputInt<=100)
                    then Zoom:=InputInt/10;
                    ch:=' ';
                  end;
          'V' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter Vertical Offset (-10 to
10 screen size): ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= -1000)
and (InputInt<1000)
```

```
                    then VertOffset:=InputInt;
                    ch:=' ';
                  end;
          'H' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Enter Horizontal Offset (-10
to 10 screen size): ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
                    if (ValCode=0) and (InputInt >= -1000)
and (InputInt<1000)
                    then HoriOffset:=InputInt;
                    ch:=' ';
                  end;
          'L' : begin
                    GotoXY(1,23);
                    ClrEol;
                    GotoXY(1,24);
                    ClrEol;
                    GotoXY(1,25);
                    ClrEol;
                    GotoXY(1,24);
                    write('   Length =',normlength,' Enter
new normal factor: ');
                    CursorOn;
                    Readln(InputStr);
                    CursorOff;
                    val(InputStr,InputInt,ValCode);
```

```
                if (ValCode=0) and (InputInt >= 0) and
(InputInt<30000)
                then NormLength:=InputInt;
                ch:=' ';
              end;
        'B' : begin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
                ClrEol;
                GotoXY(1,24);
                write('   Begin =',BeginFacet,' Enter
new beginning facet: ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >= 1) and
(InputInt<EndFacet)
                then BeginFacet:=InputInt;
                ch:=' ';
              end;
        'E' : begin
                GotoXY(1,23);
                ClrEol;
                GotoXY(1,24);
                ClrEol;
                GotoXY(1,25);
                ClrEol;
                GotoXY(1,24);
                write('   End =',EndFacet,' Enter new
ending facet: ');
                CursorOn;
                Readln(InputStr);
                CursorOff;
```

```
                val(InputStr,InputInt,ValCode);
                if (ValCode=0) and (InputInt >
BeginFacet) and (InputInt<=TriangleSelected+1)
                then EndFacet:=InputInt;
                ch:=' ';
              end;
        'G' : begin
                if (DisplayMode='S') and (not SSorted)
then begin
                  WSorted:=false;
                  close(TriangleFile);
                  GetTriangleInfo;
                end;
                { note that if DisplayMode<>'S' we sorted
up at the toggle so that the display on the terminal
screen is correct   }
                GraphTriangles(ch);
                ch := Tempchar;
                TempChar:= 'F';
                if (Pos(ch,'NGTWSRADZHVBEL') = 0) and
(ch<>ESC) then ch:= ' ';
                RestoreCrtMode;
              end;
      end;
    until FileError OR (ch = ESC) OR
(TriangleFilename='');
    if FileError then begin
      if LinePosValid then ASCErrorLn(TriangleFilename)
      else ASCError(TriangleFilename);
      TriangleFilename:='';
    end;
  end;



{-------------------------------------------------------
  |       Start of Main Slice Routines       |
```

```
-------------------------------------------------------}

function FindZLayer(value : Word) : Word;
  { Binary search for Z Value }
  var
    low,mid,high,index  :  Word;
  begin
    low:=0;
    high:=SliceLastLayer;
    index:=MAX_WORD;
    if (value>=SliceLayerValue^[low]) AND
(value<=SliceLayerValue^[high]) then
    while (low<=high) AND (index=MAX_WORD) do begin
      mid:=(low+high) div 2;
      if value<SliceLayerValue^[mid] then high:=mid-1
      else if value>SliceLayerValue^[mid] then
low:=mid+1
           else index:=mid;
    end;
    if index=MAX_WORD then FindZLayer:=high
    else FindZLayer:=index;
  end;


function BlockType(var line : String) : Word;
  { Returns Block Type of string in Word bits
              B        H        F
             ---      ---      ---
        L      0        1        2      (Bits Set)
        FU     3        4        5
        FD     6        7        8
        NFU    9       10       11
        NFD   12       13       14

        L(level)  = 15
        Not Found = None
   }
```

```
  var
    bitword : Word;
    i       : Word;
  begin
    if Copy(line,1,2)='L ' then bitword := L_SECTION
    else begin
      i:=0;
      while (i<MAX_SECTIONS) AND
            (Copy(line,1,Length(SliceBlocks[i]))<>
SliceBlocks[i]) do Inc(i);
      if (i=MAX_SECTIONS) OR (Length(line)=Length
(SliceBlocks[i])) then
        bitword := $0000
      else
        case (line[Length(SliceBlocks[i])+1]) of
          'B' : bitword:=1 shl (i*3);
          'H' : bitword:=1 shl (i*3+1);
          'F' : bitword:=1 shl (i*3+2);
          else  bitword:=$0000;
        end;
    end;
    BlockType:=bitword;
  end;

function BitOn(num : Word; bit : Byte) : Boolean;
  { Tells if a bit is on in number }
  begin
    if ((1 shl bit)and num)>0 then BitOn:=TRUE
    else BitOn:=FALSE;
  end;

procedure PrintWordRange(Str1,Str2 : String;min,max :
Word;control : LongInt);
  { Prints Range values on line for parameters }
  begin
    GotoXY(3,WhereY);
    TextColor(LightGray);
```

```
    write(Copy(Str1+'              ',1,12),': ');
    TextColor(White);
    if control < 0 then write('Undetermined')
    else write(min);
    GotoXY(53,WhereY);
    TextColor(LightGray);
    write(Copy(Str2+'              ',1,12),': ');
    TextColor(White);
    if control < 0 then write('Undetermined')
    else write(max);
  end;

procedure PrintBlockTable(section : Word);
  { Prints the Block table for menu from section value
at current location }
  var
    XPos,YPos   : Byte;
    i,j         : Word;
  begin
    XPos:=WhereX;
    YPos:=WhereY;
    for i:= 0 to 4 do begin
      GotoXY(XPos,YPos+i);
      TextColor(LightGray);
      write(SliceBlocks[i]);
      GotoXY(XPos+4,WhereY);
      for j:= 0 to 2 do begin
        if BitOn(section,i*3+j) then TextColor(White)
        else if Not MonoDisplay then TextColor(Blue)
             else TextColor(Black);
        if ((i <> 0) or (j <> 2)) then case j of
          0 : write(' B');
          1 : write(' H');
          2 : write(' F');
        end;
      end;
    end;
```

```
    TextColor(White);
  end;

procedure PrintSliceMenu;
  { Prints Slice Menu Screen with all information }
  begin
    PrintSliceHeader;
    GotoXY(3,5);
    TextColor(LightGray);
    write('Slice Filename : ');
    TextColor(White);
    write(SliceFilename);
    GotoXY(3,6);
    PrintWordRange('Layer Start','Layer
End',SliceLayerValue^[0],SliceLayerValue^[SliceLast
Layer],1);
    GotoXY(3,7);
    PrintWordRange('File X Min','File X
Max',SliceFileXMin,SliceFileXMax,1);
    GotoXY(3,8);
    PrintWordRange('File Y Min','File Y
Max',SliceFileYMin,SliceFileYMax,1);
    GotoXY(3,10);
    TextColor(LightGray);
    write('Selected Layer : ');
    TextColor(White);
    if SliceLayer=MAX_WORD then write('None')
    else begin
      write(SliceLayerValue^[SliceLayer]);
      GotoXY(3,11);
      PrintWordRange('Layer X Min','Layer X
Max',SliceLayerXMin^[SliceLayer],SliceLayerXMax^[SliceLa
yer],SliceLayer);
      GotoXY(3,12);
      PrintWordRange('Layer Y Min','Layer Y
Max',SliceLayerYMin^[SliceLayer],SliceLayerYMax^[SliceLa
yer],SliceLayer);
```

```
    end;
    GotoXY(3,14);
    TextColor(LightGray);
    write('Windowing by ');
    TextColor(White);
    if SliceWindowingFile then write('File')
    else write('Layer');
    TextColor(LightGray);
    GotoXY(3,16);
    write('Layer Blocks:');
    GotoXY(53,16);
    write('Display Blocks:');
    if SliceLayer<MAX_WORD then begin
      GotoXY(5,17);
      PrintBlockTable(SliceLayerBlocks^[SliceLayer]);
      GotoXY(54,17);
      PrintBlockTable(SliceOutputBlocks);
    end;
    GotoXY(1,23);
    TextColor(White);
    writeln('   Select:  N  new file          L  layer
select   G  graph  ');
    write  ('            W  windowing mode    B  block
select             ');
  end;

  function ParseSegment(line : String;var x1,y1,x2,y2 :
Word) : Boolean;
    { Parses out a line that should contain 4 segment
coords. }
    var
      try           : Integer;
      num           : Word;
      good          : Boolean;
      code          : Word;
      oldlen        : Char;
      j             : Word;
```

```
    begin
      good:=TRUE;
      try:=1;
      repeat
        if line[0]=#0 then good:=FALSE
        else begin
          if line[1]=' ' then line[1]:='0';
          j:=Pos(' ',line);
          if j=0 then j:=Length(line)+1;
          oldlen := line[0];
          line[0] := Char (j-1);
          Val(line,num,code);
          line[0] := oldlen;
          if code>0 then good:=FALSE
          else case (try) of
            1 : x1:=num;
            2 : y1:=num;
            3 : x2:=num;
            4 : y2:=num;
          end;
          if j>=Length(line) then line:=''
          else begin
            line[0] := Char (Length(line)-j);
            Move(line[j+1],line[1],Integer (line[0]));
          end;
        end;
        Inc(try);
      until (Not good) OR (try>4);
      ParseSegment := good;
    end;

  procedure GetSliceInfo;
    { Reads a Slice File and sets up Range and Layer
information }
    var
      sectype                : Word;
      x1,y1,x2,y2            : Word;
```

```
    begin
      SliceFileXMin  := MAX_WORD;
      SliceFileXMax  := 0;
      SliceFileYMin  := MAX_WORD;
      SliceFileYMax  := 0;
      SliceLayer     := MAX_WORD;
      SliceLastLayer := -1;
      Assign(SliceFile,SliceFilename);
      Reset(SliceFile,1);
      ResetFile(SliceFile);
      ReadLine(SliceFile);
      repeat
        if (Not EndOfFile) AND (InputLine[1]<>'!') then
begin
          sectype := BlockType(InputLine);
          if sectype = 0 then
            if (Not ParseSegment(InputLine,x1,y1,x2,y2))
or (SliceLastLayer<0) then FileError:=TRUE
            else begin
              { A Segment }
              SetMinMax(x1,SliceLayerXMin^
[SliceLastLayer],SliceLayerXMax^[SliceLastLayer]);
              SetMinMax(x2,SliceLayerXMin^
[SliceLastLayer],SliceLayerXMax^[SliceLastLayer]);
              SetMinMax(y1,SliceLayerYMin^
[SliceLastLayer],SliceLayerYMax^[SliceLastLayer]);
              SetMinMax(y2,SliceLayerYMin^
[SliceLastLayer],SliceLayerYMax^[SliceLastLayer]);
            end
          else
            { A Block line }
            if sectype=L_SECTION then begin
              if SliceLastLayer>=0 then begin
                if SliceLayerXMin^[SliceLastLayer]
<SliceFileXMin then SliceFileXMin:=SliceLayerXMin^
[SliceLastLayer];
```

```
                if SliceLayerXMax^[SliceLastLayer]>
SliceFileXMax then SliceFileXMax:=SliceLayerXMax^
[SliceLastLayer];
                if SliceLayerYMin^[SliceLastLayer]
<SliceFileYMin then SliceFileYMin:=SliceLayerYMin^
[SliceLastLayer];
                if SliceLayerYMax^[SliceLastLayer]>
SliceFileYMax then SliceFileYMax:=SliceLayerYMax^
[SliceLastLayer];
              end;
              Inc(SliceLastLayer);
              if SliceLastLayer>Total_Layers then begin
                Str(SliceLastLayer,TempStr);
                Center('Warning - Maximum number of
'+TempStr+' Slice layers read',24);
                WaitKey;
              end else begin
                Val(Copy(InputLine,3,255),TempWord,rc);
                if rc<>0 then FileError:=TRUE
                else begin
                  SliceLayerValue^[SliceLastLayer]:=
TempWord;
                  SliceLayerBlocks^[SliceLastLayer]:=
$0000;
                  SliceLayerPtr^[SliceLastLayer]:=
CurrFilePos;
                  SliceLayerXMin^[SliceLastLayer]:=
MAX_WORD;
                  SliceLayerXMax^[SliceLastLayer]:=0;
                  SliceLayerYMin^[SliceLastLayer]:=
MAX_WORD;
                  SliceLayerYMax^[SliceLastLayer]:=0;
                end;
              end;
            end else SliceLayerBlocks^[SliceLastLayer]:=
SliceLayerBlocks^[SliceLastLayer] or sectype;
        end;
```

```
        ReadLine(SliceFile);
      until FileError OR EndOfFile OR
(SliceLastLayer>Total_Layers);
      if Not FileError then begin
        if SliceLastLayer>Total_Layers then
SliceLastLayer:=Total_Layers;
        if SliceLayerXMin^[SliceLastLayer]<SliceFileXMin
then SliceFileXMin:=SliceLayerXMin^[SliceLastLayer];
        if SliceLayerXMax^[SliceLastLayer]>SliceFileXMax
then SliceFileXMax:=SliceLayerXMax^[SliceLastLayer];
        if SliceLayerYMin^[SliceLastLayer]<SliceFileYMin
then SliceFileYMin:=SliceLayerYMin^[SliceLastLayer];
        if SliceLayerYMax^[SliceLastLayer]>SliceFileYMax
then SliceFileYMax:=SliceLayerYMax^[SliceLastLayer];
      end;
    end;

procedure GraphSliceLayer;
  var
    sectype,currentsection : Word;
    x1,y1,x2,y2            : Word;
  begin
    if SliceWindowingFile then
      SetupGraphicW(SliceFileXMin,SliceFileXMax,
SliceFileYMin,SliceFileYMax)
    else
      SetupGraphicW(SliceLayerXMin^[SliceLayer],
SliceLayerXMax^[SliceLayer],
                    SliceLayerYMin^[SliceLayer],
SliceLayerYMax^[SliceLayer]);
    OutTextLn('Filename: '+SliceFIlename);
    Str(SliceLayerValue^[SliceLayer],TempStr);
    OutTextLn('Layer   : '+TempStr);
    GrTextXY('press Q to quit display-',43,1);
    GrTextXY('or <esc> for main menu',43,2);
```

```
    PositionFile(SliceFile,SliceLayerPtr^[SliceLayer]);
    repeat
      ReadLine(SliceFile);
      if (Not EndOfFile) AND (InputLine[1]<>'!') then
begin
        sectype := BlockType(InputLine);
        if sectype=0 then begin
          { A Segment }
          if (currentsection>0) AND ParseSegment
(InputLine,x1,y1,x2,y2) then DrawLine(x1,y1,x2,y2);
        end else begin
          { A Block line }
          currentsection:=sectype AND SliceOutputBlocks;
          if currentsection>0 then
            if currentsection=LB_SECTION then
SelectGraphColor(Blue)
            else if currentsection=LH_SECTION then
SelectGraphColor(LightBlue)
            else if (currentsection and NFD_SECTIONS)>0
then SelectGraphColor(Red)
            else if (currentsection and NFU_SECTIONS)>0
then SelectGraphColor(Green)
            else if (currentsection and FD_SECTIONS)>0
then SelectGraphColor(LightRed)
            else if (currentsection and FU_SECTIONS)>0
then SelectGraphColor(LightGreen)
            else SelectGraphColor(White);
        end;
      end;
    until (sectype=L_SECTION) OR EndOfFile;
  end;


procedure DoSlice;
  { User selected Slice option of Main Menu }
  var
    ch  : Char;
```

```
    i,j : Word;
    index : Integer;
  begin
    ch := ' ';
    repeat
      if SliceFilename = '' then ch := 'N'
      else begin
        ResetFile(SliceFile);
        PrintSliceMenu;
        if ch=' ' then repeat
          ch:= UpCase(ReadKey);
        until (ch = ESC) or (Pos(ch,'NLBWG') > 0);
        TextColor(White);
      end;
      LinePosValid:=FALSE;
      case ch of
        'N' : begin
                if GetFilename(SliceFilename,2) then
begin
                  Center('<Scanning Slice File>',25);
                  GetSliceInfo;
                  LinePosValid:=TRUE;
                  SliceWindowingFile:=TRUE;
                  SliceLayer:=0;
                  SliceOutputBlocks:=SliceLayerBlocks^
[SliceLayer];
                end;
                ch:=' ';
              end;
        'L' : begin
                index:=0;
                TempInt:=1;
                repeat
                  if TempInt<>index then begin
                    ClrScr;
                    Center('Slice Layer Select',1);
```

```
                    for j:=0 to 15 do if j+index<=
SliceLastLayer then begin
                      GotoXY(1+(j mod 4)*20,3+(j div
4)*5);
                      write(SliceLayerValue^[j+index]:
5,' ');
                      PrintBlockTable(SliceLayerBlocks^
[j+index]);
                    end;
                    Center('PgUp or PgDn to move
backward or forward. ENTER to select Level',24);
                    TempInt:=index;
                  end;
                  ch:=ReadKey;
                  if ch=#0 then begin
                    ch:=ReadKey;
                    if (ch=#81) and
(index+16<=SliceLastLayer) then Inc(index,16);
                    if (ch=#73) and (index-16>=0) then
Dec(index,16);
                  end;
                until ch=CR;
                GotoXY(1,24);
                ClrEol;
                write('Enter Layer Number: ');
                readln(TempStr);
                if TempStr<>'' then begin
                  Val(TempStr,iw,rc);
                  jw:=FindZLayer(iw);
                  if (rc<>0) OR (jw=MAX_WORD) then begin
                    Center('Layer "'+TempStr+'" Not
Found',24);
                    WaitKey;
                  end else begin
                    SliceLayer:=jw;
                    SliceOutputBlocks:=SliceLayerBlocks^
[SliceLayer];
```

```
                    end;
                  end;
                  ch:=' ';
                end;
          'B' : begin
                  GotoXY(1,24);
                  ClrEol;
                  write('Enter Block Type to Toggle: ');
                  readln(TempStr);
                  ToUpper(TempStr);
                  SliceOutputBlocks:=SliceOutputBlocks xor
BlockType(TempStr);
                  ch:=' ';
                end;
          'W' : begin
                  SliceWindowingFile := not
SliceWindowingFile;
                  ch:=' ';
                end;
          'G' : if SliceLayer<MAX_WORD then begin
                  repeat
                    GraphSliceLayer;
                    ch:=UpCase(ReadKey);
                  until ch<>'G';
                  RestoreCRTMode;
                end else ch:=' ';
          ELSE if ch<>ESC then ch:=' ';
        end;
      until (ch = ESC) OR (SliceFilename='');
      if FileError then begin
        if LinePosValid then ASCErrorLn(TriangleFilename)
        else ASCError(TriangleFilename);
        TriangleFilename:='';
      end;
    end;
```

```
{-------------------------------------------------------
  |                Start of Main Program                |

-------------------------------------------------------}

  begin
    { Setup Exit routine }
    ExitSave:=ExitProc;
    ExitProc:=@MainExit;

    { Determine graphics screen to use }
    GraphDriver := Detect;
    SetGraph;
    RestoreCrtMode;
{    assign(gftest,'gftest.txt');
    rewrite(gftest); }

    { Use this heap location as top of heap for rest of
program }
    Mark(HeapTop);

    { Allocate Memory for arrays }
    ReAllocate('S',16382);
    if Total_Triangles<5000 then ReAllocate('T',5000);

    repeat
      PrintHeader;
      Center('Graphic Card:
'+GraphDriverName(GraphDriver),WhereY+2);
      Str(GraphMode,TempStr1);
      TempStr:='Graphics Mode:'+TempStr1+' (';
      Str(SizeX,TempStr1);
      TempStr:=TempStr+TempStr1+',';
      Str(SizeY,TempStr1);
      TempStr:=TempStr+TempStr1+')';
      Center(TempStr,WhereY+1);
```

```
        GotoXY(1,25);
        write('Maximum Triangles: ',Total_Triangles+1);
        GotoXY(57,25);
        write('Maximum Layers: ',Total_Layers+1);


        GotoXY(22,10);
        write('1. Graph Triangles ( .STL file )');
        GotoXY(22,WhereY+2);
        write('2. Graph Slice Layers ( .SLI file )');
        GotoXY(22,WhereY+2);
        write('3. Change Triangles / Layer Allocation');
        GotoXY(22,WhereY+2);
        write('Q. Quit program');

        GotoXY(28,WhereY+3);
        write('Enter Selection: ');
        repeat
          ch := UpCase(ReadKey);
        until Pos(ch,'123Q')>0;
        Case ch of
          '1' : if Total_Triangles>=0 then DoTriangle;
          '2' : if Total_Layers>=0 then DoSlice;
          '3' : begin
                  ClrScr;
                  GotoXY(1,24);
                  write('Enter number of Slice Layers
desired: ');
                  readln(TempStr);
                  if TempStr<>'' then begin
                    Val(TempStr,TempReal,rc);
                    TempLong:=Round(TempReal);
                    if (rc=0) and (TempLong>=0) then
ReAllocate('S',TempLong);
                  end;
                end;
        end;
```

```
    until (ch = 'Q');
    TextBackground(Black);
    TextColor(White);
{    close(gftest); }
    CursorOn;
    ClrScr;
  end.
```

3.20 SOURCE .INC FILES

November 07, 1988

beamcali.inc
beamgr.inc
bench.inc
bldmon.inc
bdlzstg.inc
lctrl.inc
beamsave.inc
bldfile.inc
bldopt.inc
buildgr.inc
newview.inc
bldsweep.inc
beamio.inc
bldgr.inc
bldstat.inc
combine.inc
stercmds.inc
beamscan.inc
bldkbd.inc
bldtime.inc
graph.inc
view.inc
beamutil.inc
bldlaser.inc
bldutil.inc
laser.inc
bldfile1.inc.

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================

   BEAMCALI.INC
   SLA-1 Calibration Process Driver for the BEAM program

---------------------------------------------------------
procedure UserWait;
begin
   writeln;
   writeln;
   writeln('       ... press any key to continue ');
   repeat delay(100) until KeyPressed;
   junkkey:=Readkey;
end;

{
   ******************************
   * Option 8: Calibrate System *
   ******************************
}
procedure Calibr8;
const
  CalibHoleSeparation = 890;
  DEFAULTchs    = 890;  { 3556 vat units per inch * 1/4
inch = 889 }
  XVatCenter    = 32767;
  YVatCenter    = 32767;
  {   s = calibration hole separation }
  NormSpiralDelay = 100;
  NormColsDelay   = 10;
  NormRowsDelay   = 60;
var
  CalibKey : Char;
  i,code,CalibAttempts: Integer;
  tableX,tableY,FullPart,ADC: Real;
```

```
  RowStartXsp,RowStartYsp: LongInt;
  row,col,Xcount,Ycount: Integer;
  CalibCoords: array[1..50,1..50,1..2] of Real;
  CalibTimeout,GoodRow,GoodCol,ALLHOLESOK: Boolean;
  SpiralDelay: Word;  { in milliseconds }
  BetweenRowsDelay,BetweenColsDelay: Word;  { in tenths
of seconds }
  temp,Indent: String;
  passOption,passReturnC: integer;
  passColX,passRowY : integer;
  X1,Y1,X25,Y25,oldXsp,oldYsp : longint;
  Xchs1,Xchs5,Xchs10,Xchs15,Xchs20,{Xchs25 is default}
                        Xchs30,Xchs35,Xchs40,Xchs45,
Xchs49:integer;
  Ychs1,Ychs5,Ychs10,Ychs15,Ychs20,{Ychs25 is default}
                        Ychs30,Ychs35,Ychs40,Ychs45,
Ychs49:integer;

{  label
  AbortCalib; }
  label
  RePCP;  { Repeat Process Calibration PinHoles }

procedure CalibrationDelay(time{in 1/10 secs}:LongInt);
var
  sec,TotalSec: LongInt;
begin
  if time > 0 then begin
    delay(999);    { keep the calibration hole message
up for a second }
    ClearLine(23);
    gotoxy(33,23);
    sec := 2;      { start at two seconds to account for
first delay   }
    TotalSec := time div 10;
    if TotalSec > 0 then write('Waiting ...');
    while (sec <= TotalSec) and not keypressed do begin
```

```
        gotoxy(45,23);
        write(TotalSec + 1 - sec,' ');
        delay(999);
        inc(sec);
      end;
      delay(100 * time mod 10);
      while keypressed do key := ReadKey;
      ClearLine(23);
    end;
  end;

function StartCalibFile: Boolean;
begin
  StartCalibFile := true;
  assign(CalOutFile,CalibFileName);
  {$I-}
  rewrite(CalOutFile);
  if IOresult<>0 then begin
    Center('Cannot write to Calibration File',24);
    WaitKey;
    StartCalibFile := false;
    exit;
  end;
  {$I+}
  writeln(CalOutFile,'! Calibration File V' + VersionId
+ ', started ',TimeStamp);
  close(CalOutFile);
end;

function UpdateCalibFileRow: Boolean;
var
  i: Integer;
  FoundBeam1,FoundBeam2: Boolean;
  SensorPosX,SensorPosY: array[1..6] of Real;
begin
  UpdateCalibFileRow := true;
  assign(CalOutFile,CalibFileName);
```

```
  {$I-}
  append(CalOutFile);
  if IOresult<>0 then begin
    Center('Error writing to Calibration File',24);
    WaitKey;
    UpdateCalibFileRow := false;
    exit;
  end;
  {$I+}
  MoveBeam(trunc(Xsp[1]),trunc(Ysp[1]));
  CalibrationDelay(BetweenRowsDelay);
  SetupFindBeam(FindBeamVariables,1{Sensor},-1
{Xsp[1]},-1{Ysp[1]},40{SpiralSideMax},
    1{SpiralTries},4{ScanStep},trunc(Bk[1]+0.5)
{BkgndThreshold},
    20{SpiralDelay},1{NumCenterPasses});
  FoundBeam1 := FindBeam(FindBeamVariables,1,
NormalFind,0{no mssgs});
  SensorPosX[1] := FindBeamVariables[1].Xavg;
  SensorPosY[1] := FindBeamVariables[1].Yavg;
  MoveBeam(trunc(Xsp[2]),trunc(Ysp[2]));
  CalibrationDelay(BetweenRowsDelay);
  SetupFindBeam(FindBeamVariables,2
{Sensor},-1{Xsp[2]},-1{Ysp[2]},40{SpiralSideMax},
    1{SpiralTries},4{ScanStep},trunc(Bk[2]+0.5)
{BkgndThreshold},
    20{SpiralDelay},1{NumCenterPasses});
  FoundBeam2 := FindBeam(FindBeamVariables,2,NormalFind,
0{no mssgs});
  SensorPosX[2] := FindBeamVariables[2].Xavg;
  SensorPosY[2] := FindBeamVariables[2].Yavg;
  for i:=3 to 6 do begin
    SensorPosX[i] := 0;
    SensorPosY[i] := 0;
  end;
  if not (FoundBeam1 and FoundBeam2) then write
(CalOutFile,'T ');
```

```
  for i:=1 to 6 do begin
    if i > 1 then write(CalOutFile,',');
    write(CalOutFile,RealStr(SensorPosX[i],2),',',
      RealStr(SensorPosY[i],2));
  end;
  writeln(CalOutFile);
  close(CalOutFile);
end;

function UpdateCalibFileCol: Boolean;
var
  Resist,Temper: Real;
begin
  UpdateCalibFileCol := true;
  ReadTemperature(Resist,Temper);
  assign(CalOutFile,CalibFileName);
  {$I-}
  append(CalOutFile);
  if IOresult<>0 then begin
    Center('Error writing to Calibration File',24);
    WaitKey;
    UpdateCalibFileCol := false;
    exit;
  end;
  {$I+}
  GetTimeStamp;
  if CalibTimeout then write(CalOutFile,'T ');
  writeln(CalOutFile,RealStr(Xsp[Sensor],2),',',
RealStr(Ysp[Sensor],2),',',
    RealStr(Power[Sensor],2),',',RealStr(Temper,2),',',
CurrentTime);
  close(CalOutFile);
  CalibTimeout := false;
end;

function QuickCalibSearch: Boolean;
begin
```

```
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    40{SpiralSideMax},1{SpiralTries},16{ScanStep},trunc
(Bk[Sensor]+0.5),
    SpiralDelay,3{NumCenterPasses});
  if SpiralSearchForHole(2,FindBeamVariables,Sensor)
then begin
    QuickCalibSearch := true;
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end else QuickCalibSearch := false;
end;

function CalibFind: Boolean;
{ CalibFind starts its search at Xsp[Sensor],
Ysp[Sensor], and
  resets these values when the beam is found. }
begin
    SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    40{SpiralSideMax},3{SpiralTries},9{ScanStep},trunc
(Bk[Sensor]+0.5),
    SpiralDelay,2{NumCenterPasses});
  if FindBeam(FindBeamVariables,Sensor,NormalFind,2{no
mssgs,noverify}) then begin
    CalibFind := true;
    Xsp[Sensor] :=
trunc(FindBeamVariables[Sensor].Xavg+0.5);
    Ysp[Sensor] :=
trunc(FindBeamVariables[Sensor].Yavg+0.5);
  end else CalibFind := false;
end;

procedure UserWait;
begin
   writeln;
   writeln;
```

```
   writeln('          ... press any key to continue ');
   repeat delay(100) until KeyPressed;
   junkkey:=Readkey;
end;

procedure
GetBackgroundNoiseLevels(Xcoord,Ycoord:longint);
begin
  with BeamProfileVariables[Sensor] do begin
    SensorBkgnd[Sensor]  :=
BkReadFunction(Sensor,Xcoord,Ycoord);
                              { true background }
    BkgndThreshold       := trunc(Bk[Sensor]+0.5);
{ high level mark }
    { Bk[Sensor] is used by FindBeamVariables }
  end;
end;

procedure CenterCalibPlate;
var
   i : integer;
   CENTERED : boolean;
begin
   Xsp[Sensor] := XVatCenter;
   Ysp[Sensor] := YVatCenter;
   MoveBeam(XVatCenter,YVatCenter);
   ClrScr;
   gotoxy(1,8);
   writeln('     1. The beam is positioned in the middle
of the vat (32767,32767).');
   writeln;
   writeln('     2. Center the calibration plate by the
using beam position.');
   writeln('        Position plate with mirrored section
centered at 7.25" x 7.25".');
   writeln;
```

```
   writeln('     3. Find the actual coordinates of the
center pinhole by pressing');
   writeln('        ENTER to begin the search.');
   writeln;
   writeln('        ( or press X to exit )');
   repeat delay(100) until KeyPressed;
   junkkey := readkey;
   if upcase(junkkey) = 'X' then begin
      BORDERSOK := false;
      exit;
   end;
   writeln;
   writeln('     Please wait while the centerhole is
located. ');

   CENTERED := false;
   while not CENTERED do begin
   Xsp[Sensor] := XVatCenter;
   Ysp[Sensor] := YVatCenter;
   MoveBeam(XVatCenter,YVatCenter);
   if CalibFind then begin
      ClrScr;
      gotoxy(1,9);
      writeln('   ***  Calibration plate center hole
located at: ',Xsp[Sensor]:5:0,',',Ysp[Sensor]:5:0);
      writeln;
      if (Abs(32767-Xsp[Sensor]) < 80) and
(Abs(32767-Ysp[Sensor]) < 80) then begin
         writeln('   These values are within the
acceptable tolerances for the calibration');
         writeln('   plate center hole.');
         CENTERED := true;
         userwait;
      end else begin
         writeln('   These values are not withing the
acceptable tolerances for the ');
```

```
            writeln('   calibration plate center hole
(32767 +/- 80 for X and Y).');
            writeln;
            writeln('   Take the following corrective
action(s):');
            if (32767-Xsp[Sensor] > 80) then
            writeln('      X coordinate too small - slide
plate to the left. ');
            if (Xsp[Sensor]-32767 > 80) then
            writeln('      X coordinate too large - slide
plate to the right.');
            if (Ysp[Sensor]-32767 > 80) then
            writeln('      Y coordinate too large - slide
plate forward.');
            if (32767-Ysp[Sensor] > 80) then
            writeln('      Y coordinate too small - slide
plate backwards.');
            writeln;
            writeln('      ( or press X to exit )');
            userwait;
            if upcase(junkkey)='X' then begin
               BORDERSOK := false;
               exit;
            end;
            writeln;
            writeln('     Please wait while the centerhole
is relocated. ');
         end;
      end else begin
         writeln('*****  Center Hole not found.  Reposition
plate and/or adjust gain. *****');
         writeln('       ( or press X to exit ) ');
         userwait;
         if upcase(junkkey)='X' then begin
            BORDERSOK := false;
            exit;
         end;
```

```
      writeln;
      writeln('     Please wait while the centerhole is
relocated. ');
   end;
  end; { while not CENTERED }
   X25 := trunc(Xsp[Sensor]);   { positions for
hole(25,25) }
   Y25 := trunc(Ysp[Sensor]);
end; { procedure CenterCalibPlate }

procedure AdjustGain(keychar:char);
{ 'C' is center hole, otherwise do not reset X25 and
Y25. }
var
   key: Char;
   v,code : integer;
   leave, newsensor, goodgain : boolean;
begin
   goodgain := false;
   ClrScr;
   gotoxy(1,9);
   writeln('     The next step is adjusting the gain on
the calibration plate:');
   writeln;
   writeln('        Raise the gain if the MAXIMUM
reading is below 50.');
   writeln('        Lower the gain if the MAXIMUM
reading is above 70.');
   writeln;
   writeln('        Press any key to leave the profiler
after the gain is set.');
   userwait;
   OpMode := GraphicsOp;
   Sensor := 3;
   Display := true;
   repeat
     Leave := false;
```

```
      TextMode(C80);
      SetScreen;
      gotoxy(1,25);
      write(' GATHERING PROFILE for Sensor 3 (Calibration
Plate), press ''X'' to exit');
      NewSensor := false;
     repeat
       Display := true;
       BeamProfileVariables[Sensor].ScanStep := 2;
       GoodRead(1);
       if TotWtAvg=0 then begin
         GoodGain := false;
         Center(' ERROR: Hole not located: Adjust gain.
(''X'' to exit)',24);
         WaitKey;
         junkkey := upcase(Readkey);
         if junkkey = 'X' then begin
            Display := false;
            TextMode(C80);
            ClrScr;
            exit;
         end;
       end;
       if GoodCount > 0 then begin
         GoodGain := true;
         Xsp[Sensor] := Xaverage;
         Ysp[Sensor] := Yaverage;
         if keychar = 'C' then begin
            X25 := trunc(Xsp[Sensor]);
            Y25 := trunc(Ysp[Sensor]);
         end;
       end;
       PrintInfo(Sensor);
       if keypressed then begin
         key := ReadKey;
         if upcase(key) = 'Q' then exit;
         v := IntVal(key,code);
```

```
        if (v > 0) and (v <= NumSensors) then begin
          Sensor := v;
          MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp
[Sensor]));
          delay(200);
          NewSensor := true;
        end else Leave := true;
      end;
    until NewSensor or Leave;
  until Leave and GoodGain;
  Display := false;
  TextMode(C80);
  ClrScr;
end; { procedure AdjustGain }

procedure PutCalibHoleSeparation(XHoleNum,YHoleNum,
XchsLast,YchsLast,
             NumSteps:integer; Axis:char; var
XchsNew,YchsNew:integer);
{ Determines the calibration hole separation from the
average of NumSteps holes for one axis movement.      }
var
   SaveX,SaveY: LongInt;  { coordinates of pinhole }
   SumXchs,SumYchs,i : integer;
   AXISX : boolean;
begin
   if not BORDERSOK then exit;
   SumXchs := 0;
   SumYchs := 0;
   XchsNew := XchsLast;
   YchsNew := YchsLast;
   SaveX   := trunc(Xsp[Sensor]);
   SaveY   := trunc(Ysp[Sensor]);
   if Axis = 'X' then AXISX := TRUE else AXISX := FALSE;
   for i := 1 to NumSteps do begin
      if (XHoleNum < 25) or (YHoleNum < 25) then i :=
-i;
```

```
      if AXISX then begin
         write('     Pass for hole
',XHoleNum+i,',',YHoleNum,':');
         SaveX := trunc(Xsp[Sensor]);
      end else begin
         write('     Pass for hole
',XHoleNum,',',YHoleNum+i,':');
         saveY := trunc(Ysp[Sensor]);
      end;
      if i < 0 then begin { stepping down towards 1,1 }
         Xsp[Sensor] := SaveX - XchsNew;
         Ysp[Sensor] := SaveY - YchsNew;
      end else begin
         Xsp[Sensor] := SaveX + XchsNew;
         Ysp[Sensor] := SaveY + YchsNew;
      end;
      if CalibFind then begin
         if not BORDERSOK then exit;
         writeln('   FOUND at
',Xsp[Sensor]:6:0,',',Ysp[Sensor]:6:0);
         if AXISX then begin
            XchsNew := abs(trunc(Xsp[Sensor] - SaveX));
            SumXchs := SumXchs + XchsNew;
         end else begin
            YchsNew := abs(trunc(Ysp[Sensor] - SaveY));
            SumYchs := SumYchs + YchsNew;
         end;
      end else begin
         writeln;
         writeln('     *** ERROR : calibration plate
hole not found');
         writeln('         Borders of plate are not
established.   ');
         BORDERSOK := false;
         userwait;
         writeln('     Please wait... ');
         exit;
```

```
      end;
      i := abs(i);
   end;
   if AXISX then XchsNew := abs(SumXchs div NumSteps)
            else YchsNew := abs(SumYchs div NumSteps);
end; { procedure PutCalibHoleSeparation }

procedure MakeRoughGeoCorrect;
   { X and Y coordinates for center hole 25,25 are
established.  Now establish coordinates for the
remainder of the holes in increments of 5. }
var
   junk : integer;
begin
   Xchs30 := 0;
   Ychs30 := 0;
   ClrScr;
   writeln;
   writeln('   Establish borders of calibration
plate:');
   writeln;

   { The beam is at the center hole from procedure
adjust gain. }
   { X walk up border }
   PutCalibHoleSeparation(25,25,DEFAULTchs,0,5,'X',
Xchs30,junk);
   PutCalibHoleSeparation(30,25,
Xchs30,0,5,'X',Xchs35,junk);
   PutCalibHoleSeparation(35,25,
Xchs35,0,5,'X',Xchs40,junk);
   PutCalibHoleSeparation(40,25,
Xchs40,0,5,'X',Xchs45,junk);
   PutCalibHoleSeparation(45,25,
Xchs45,0,4,'X',Xchs49,junk);
   if not BORDERSOK then exit;
```

```
   Xsp[Sensor] := Xsp[Sensor] + Xchs49; { one past
border }
   writeln;
   if CalibFind and BORDERSOK then begin
      BORDERSOK := false;
      writeln('   *** ERROR : Cannot determine X border.
');
      writeln('               Check calibration plate
positioning. ');
      userwait;
      exit;
   end else writeln(' X border has been established. ');

   { Y walk up to border on the left side }
   Xsp[Sensor] := X25;
   Ysp[Sensor] := Y25;
   PutCalibHoleSeparation(25,25,0,DEFAULTchs,5,'Y',
junk,Ychs30);
   PutCalibHoleSeparation(25,30,0,
Ychs30,5,'Y',junk,Ychs35);
   PutCalibHoleSeparation(25,35,0,
Ychs35,5,'Y',junk,Ychs40);
   PutCalibHoleSeparation(25,40,0,
Ychs40,5,'Y',junk,Ychs45);
   PutCalibHoleSeparation(25,45,0,
Ychs45,4,'Y',junk,Ychs49);
   if not BORDERSOK then exit;
   Ysp[Sensor] := Ysp[Sensor] + Ychs49; { one past
border }
   writeln;
   if CalibFind and BORDERSOK then begin
      BORDERSOK := false;
      writeln(' *** ERROR : Cannot determine Y
border. ');
      writeln(' Check calibration plate positioning. ');
      userwait;
      exit;
```

```
   end else writeln(' Y border has been established. ');

   { X walk down to the front border }
   Xsp[Sensor] := X25 - Xchs30;
   Ysp[Sensor] := Y25;
   PutCalibHoleSeparation(24,25,DEFAULTchs,0,4,'X',
Xchs20,junk);
   PutCalibHoleSeparation(20,25,
Xchs20,0,5,'X',Xchs15,junk);
   PutCalibHoleSeparation(15,25,
Xchs15,0,5,'X',Xchs10,junk);
   PutCalibHoleSeparation(10,25,    Xchs10,0,5,'X',
Xchs5,junk);
   PutCalibHoleSeparation( 5,25,     Xchs5,0,4,'X',
Xchs1,junk);

   { Y walk down to the right side : NOTE - X is not
reset to center }
   Ysp[Sensor] := Y25 - Ychs30;
   PutCalibHoleSeparation(1,24,0,DEFAULTchs,4,'Y',
junk,Ychs20);
   PutCalibHoleSeparation(1,20,0,
Ychs20,5,'Y',junk,Ychs15);
   PutCalibHoleSeparation(1,15,0,
Ychs15,5,'Y',junk,Ychs10);
   PutCalibHoleSeparation(1,10,0,    Ychs10,5,'Y',junk,
Ychs5);
   PutCalibHoleSeparation(1, 5,0,     Ychs5,4,'Y',junk,
Ychs1);

   writeln;
   if BORDERSOK then begin
      X1 := trunc(Xsp[Sensor]);    { Save the
coordinates of hole 1,1. }
      Y1 := trunc(Ysp[Sensor]);
      writeln('     Borders of Calibration Plate are
established.   ');
```

```
      writeln;
   end;
end;

function GetCalibHoleSeparation(axis:char;aisle:
integer):integer;
var
   ivalue : integer;
begin
   if axis = 'X' then begin
      if      aisle < 5  then ivalue := Xchs1
      else if aisle < 10 then ivalue := Xchs5
      else if aisle < 15 then ivalue := Xchs10
      else if aisle < 20 then ivalue := Xchs15
      else if aisle < 25 then ivalue := Xchs20
      else if aisle > 45 then ivalue := Xchs49
      else if aisle > 40 then ivalue := Xchs45
      else if aisle > 35 then ivalue := Xchs40
      else if aisle > 30 then ivalue := Xchs35
      else if aisle >= 25 then ivalue := Xchs30;
   end else begin { axis = 'Y' }
      if      aisle < 5  then ivalue := Ychs1
      else if aisle < 10 then ivalue := Ychs5
      else if aisle < 15 then ivalue := Ychs10
      else if aisle < 20 then ivalue := Ychs15
      else if aisle < 25 then ivalue := Ychs20
      else if aisle > 45 then ivalue := Ychs49
      else if aisle > 40 then ivalue := Ychs45
      else if aisle > 35 then ivalue := Ychs40
      else if aisle > 30 then ivalue := Ychs35
      else if aisle >= 25 then ivalue := Ychs30;
   end;
   GetCalibHoleSeparation := ivalue;
end;  { function GetCalibHoleSeparation }

procedure ProcessCalibrationPinHoles(var
iOption,iReturn,Rowx,Colx: integer);
```

```
{  iOption = 0 is regular calibration.
           = 1 is a quick scan.
   iReturn = 0 is all holes found
           = 1 could not locate a pinhole
           = 2 pinhole ADC reading over saturation level
               iReturn <> 0 returns BadX and BadY }
{
   Calibration:

   Alg moves out each row (Y-axis), then out each column
(X-axis) looking for a hole.  The current alg looks for
a set of 49 x 49 holes.  For each position, whether it
finds a hole or not, it makes an entry in the CALIB.DAT
file.  Entries in this file for holes for which the
laser beam wasn't found after 10 attempts are preceded
by a 'T '.
}
var
   row,col : integer;
begin
  if iOption = 0 then if not StartCalibFile then exit;
  iReturn := 0;
  CalibTimeout := false;
  Xsp[Sensor] := X1; { Start at established corner hole
1,1.}
  Ysp[Sensor] := Y1;
  for row := 1 to 49 do begin
    RowX := row;
    GoodRow := CalibFind;
    if iOption = 0 then if not UpdateCalibFileRow then
exit;
    MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
    if iOption = 0 then
CalibrationDelay(BetweenRowsDelay);
    for col := 1 to 49 do begin
      colx := col;
      if AbortFlag then exit;
```

```
      oldXsp := trunc(Xsp[Sensor]);
      oldYsp := trunc(Ysp[Sensor]);
      CalibAttempts := 0;
      repeat
        GetBackgroundNoiseLevels(trunc(Xsp[Sensor])-445,
trunc(Ysp[Sensor])-445);
        { Xsp[Sensor] and Ysp[Sensor] are used to
position CalibFind. }
        GoodCol := CalibFind;

        if AbortFlag then exit;
        if not GoodCol then begin
          inc(CalibAttempts);
          beep;
          iReturn := 1;
          center('Calibration hole
('+IntStr(row)+','+IntStr(col)+
                 ') not found on attempt number :
'+IntStr(CalibAttempts)+
                 ' (3 max).',25);
          if CalibAttempts < 3 then begin
              GoodCol := false
            end else begin
              GoodCol := true; { go on after 10
attempts }
              Center('*** Calibration hole
('+IntStr(row)+','+IntStr(col)+
                 ') not found in ten attempts, search
aborted.',25);
              CalibTimeout := true;   { set calibration
timeout flag }
            end;
          Xsp[Sensor] := oldXsp;
          Ysp[Sensor] := oldYsp;
        end;
        if GoodCol then begin
          if col = 1 then begin
```

```
            RowStartXsp := trunc(Xsp[Sensor]);
            RowStartYsp := trunc(Ysp[Sensor]);
          end;
          Ysp[Sensor] := Ysp[Sensor] +
GetCalibHoleSeparation('Y',col);
        end;
      until GoodCol;
      if iOption = 0 then if not UpdateCalibFileCol then
exit;
      Center('Calibration hole ('+IntStr(row)+','+
IntStr(col)+')',25);
      if col < 49 then begin
        MoveBeam(trunc(Xsp[Sensor]),trunc(Ysp[Sensor]));
        if iOption = 0 then CalibrationDelay
(BetweenColsDelay);
      end;
      if (iOption = 1) and (iReturn = 1) then begin
          exit;
      end;
    end; {col}
    Xsp[Sensor] := RowStartXsp;
    if row < 49 then begin
      oldXsp := trunc(Xsp[Sensor]);
      oldYsp := trunc(Ysp[Sensor]);
      CalibAttempts := 0;
      repeat
        Xsp[Sensor] := Xsp[Sensor] +
GetCalibHoleSeparation('X',row);
        Ysp[Sensor] := RowStartYsp;

        GetBackgroundNoiseLevels(trunc(Xsp[Sensor])-445,
trunc(Ysp[Sensor])-445);

        { Xsp[Sensor] and Ysp[Sensor] are used to
position CalibFind. }
        GoodRow := CalibFind;
        if AbortFlag then exit;
```

```
        if not GoodRow then begin
          inc(CalibAttempts);
          beep;
          iReturn := 1;
          center('Row '+IntStr(row)+' not found on
attempt number : '
                                    +IntStr(CalibAttempts)
+'(3 max).',25);
          if CalibAttempts < 3 then GoodCol := false
            else begin
              GoodRow := true;        { go on after 3
attempts }
              CalibTimeout := true;   { set calibration
timeout flag }
            end;
          Xsp[Sensor] := oldXsp;
          Ysp[Sensor] := oldYsp;
        end;
      until GoodRow;
    end;
    if (iOption = 1) and (iReturn = 1) then begin
       exit;
    end;
  end; {row}
  if iOption = 0 then begin
     if not UpdateCalibFileRow then exit;
     Center('Calibration completed.',25);
  end;
  WaitKey;
  exit;

end; {procedure ProcessCalibrationPinHoles }

{********* BEGIN MAIN CALIBRATION PROGRAM *************}

begin
  window (1,1,80,25);
```

```
  ClrScr;
  Calibrate := true;
  BORDERSOK := true;
  GetTimeStamp;
  CalibFileName := '\CAL\' + FNTimeStamp + '.CAL';
{  Center('Beam Calibration Function',1); }
  gotoxy(1,8);
{***
  Indent := many(' ',16);
  writeln(Indent,'Type      Fast    Medium   Slow');
  writeln(Indent,'--------------   ------   ------
------');
  writeln(Indent,'Spiral
',CalibDelays[1,1]:4,'     ',
    CalibDelays[2,1]:4,' ',CalibDelays[3,1]:4,'  ms');
  writeln(Indent,'Between Points
',CalibDelays[1,2]/10:5:1,'    ',
    CalibDelays[2,2]/10:5:1,'
',CalibDelays[3,2]/10:5:1,'  secs');
  writeln(Indent,'Between Rows
',CalibDelays[1,3]/10:5:1,'    ',
    CalibDelays[2,3]/10:5:1,'
',CalibDelays[3,3]/10:5:1,'  secs');
  writeln(Indent,'       -----    -----    -----');
  writeln(Indent,'TOTAL            ',
    (2.5*10*49*49+CalibDelays[1,2]*49*49+CalibDelays
[1,3]*49*3)/36000:5:1,'    ',
    (2.5*10*49*49+CalibDelays[2,2]*49*49+CalibDelays
[2,3]*49*3)/36000:5:1,'    ',
    (2.5*10*49*49+CalibDelays[3,2]*49*49+CalibDelays
[3,3]*49*3)/36000:5:1,'  hours');
   ***}
  writeln;
  Center('Normal ('+IntStr(NormSpiralDelay)+',
'+IntStr(NormColsDelay)+', '+
          IntStr(NormRowsDelay)+') or Different
Calibration?',12);
```

```
  CalibKey := upcase(ReadKey);
  case CalibKey of
{***
    'F': begin
           SpiralDelay      := CalibDelays[1,1];    in
milliseconds
           BetweenColsDelay := CalibDelays[1,2];    in
tenths of seconds
           BetweenRowsDelay := CalibDelays[1,3];
         end;
    'M': begin
           SpiralDelay      := CalibDelays[2,1];
           BetweenColsDelay := CalibDelays[2,2];
           BetweenRowsDelay := CalibDelays[2,3];
         end;
    'S': begin
           SpiralDelay      := CalibDelays[3,1];
           BetweenColsDelay := CalibDelays[3,2];
           BetweenRowsDelay := CalibDelays[3,3];
         end;
 ***}
    'N': begin
           SpiralDelay      := NormSpiralDelay;
           BetweenColsDelay := NormColsDelay;
           BetweenRowsDelay := NormRowsDelay;
         end;
    'D': begin
           writeln;
           writeln;
           write('Enter Spiral Delay (in ms): ');
           readln(temp);
           if temp='' then exit
             else SpiralDelay := LongVal(temp,code);
           write('Enter Between Points Delay (in
secs): ');
           readln(temp);
           if temp = '' then exit
```

```
            else BetweenColsDelay :=
trunc(RealVal(temp,code)*10+0.5);
          write('Enter Between Rows Delay (in
secs): ');
          readln(temp);
          if temp = '' then exit
            else BetweenRowsDelay := trunc(RealVal
(temp,code)*10+0.5);
        end;
    else exit;
  end;
  SetScreen;
  case CalibKey of
{***
    'F': Center('Fast Beam Calibration, started
'+TimeStamp,10);
    'M': Center('Medium Beam Calibration, started
'+TimeStamp,10);
    'S': Center('Slow Beam Calibration, started
'+TimeStamp,10);
 ***}
    'N': Center('Normal Beam Calibration, started
'+TimeStamp,10);
    'D': Center('Different Beam Calibration, started
'+TimeStamp,10);
  end;
  delay(900);
{*************************************************}

  Search7;
  BORDERSOK := true;
  Display := false;
  Sensor := 3;
  with FindBeamVariables[Sensor] do begin
      TextMode(C80);
      GetBackgroundNoiseLevels(Xrest,Yrest);
      CenterCalibPlate;
```

```
      if BORDERSOK then AdjustGain('C');
      if BORDERSOK then MakeRoughGeoCorrect;
      { Establishes borders and builds separation
table }
  end;
  TextMode(C80);
  ClrScr;
  if not BORDERSOK then begin
     writeln;
     writeln('   ***** Aborting Calibration *****   ');
     userwait;
     exit;
  end;

  passOption := 1;   { quick check for pinholes }
  repeat
  RePCP:   window(1,1,80,25);
     ClrScr;
     gotoxy(1,7);
     writeln('          Calibrate: Quick search for all
pinholes');
     writeln;
     writeln('          Please wait for two hours to
verify');
     writeln('          that all pinholes in the
calibration plate ');
     writeln('          can be located and that no
readings are above');
     writeln('          the saturation level.');
     writeln;
     writeln('          ( or press X to bypass the quick
search )');
  if upcase(junkkey) <> 'X' then begin

     ProcessCalibrationPinholes(passOption,passReturnC,
passColX,passRowY);
     if passReturnC = 0 then begin
```

```
        delay(1000);
      end else begin
        ClrScr;
        if passReturnC = 1 then begin { a pinhole was
not found }
        { it could be dust or a gain level set too low }
          writeln('   ***    Pinhole
',passColX,',',passRowY,' was not found.');
          writeln('          This is probably due to
and improper gain setting,');
          writeln('          and/or dust on the
calibration plate.');
          writeln;
          writeln('          The beam is positioned
near the pinhole it could not read.');
          writeln('          Adjust the gain until the
pinhole reads correctly, or hit');
          writeln('          ''X'' to exit the adjust
gain screen if the pinhole cannot');
          writeln('          be profiled. ');
          userwait;
          AdjustGain('X');   { not the center hole }
          goto RePCP;
        end;
        if passReturnC = 2 then begin { saturated
reading received }
          writeln('   ***    Pinhole
',passColX,',',passRowY,' was saturated.');
          writeln('          Please lower the gain
control. ');
          userwait;
          goto RePCP;
        end;
      end;
  end else passReturnC := 0; { if 'X' was entered to
skip quick search }
  until passReturnC = 0;
```

```
  {***}
  window(1,1,80,25);
  ClrScr;
  writeln;
  case CalibKey of
  {***
    'F': writeln('Fast Beam Calibration, started
'+TimeStamp);
    'M': writeln('Medium Beam Calibration, started
'+TimeStamp);
    'S': writeln('Slow Beam Calibration, started
'+TimeStamp);
   ***}
    'N': writeln('     Normal Beam Calibration final
pass, started '+TimeStamp);
    'D': writeln('     Different Beam Calibration final
pass, started '+TimeStamp);
  end;
  writeln;
  passOption := 0; { normal calibration : after all
holes are found }
  ProcessCalibrationPinholes(passOption,passReturnC,
passColX,passRowY);
end;

{-----------------------------------------------------}

procedure CALIMENU;

const
  CtrlC = ^C;

type
   str80 = string[80];

var
  key: Char;
```

```
  regs: Registers;
  i, indent : integer;
  InputStr  : str80;
  FileOK    : boolean;

{   Cursor on / off routines }

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;
```

```
function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
  writeln(text);
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = Ctrl C then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;
```

```
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                 {  Type          1    2    3    4      }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
```

```
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;

procedure GetSelection;
begin
  repeat
    HandleKeyboardRequest;
    key := upcase(key);
    if key in ['1'..'5','X'] then exit;
  until false;
end;

procedure ShowCaliMenu;
var
  i : integer;
begin
  window(1,1,80,25);
  textmode(C80);
  writeln;
  writeln;
  centerln('SLA-1 Calibration Control Program');
  centerln('Version '+VersionId);
  centerln('Copyright (C) 1988 by 3D Systems, Inc.');
  centerln('3D Systems Laser Stereolithography System');

  for i:=1 to 7 do writeln;
  DrawBox(Indent-1,8,80-Indent,20,1);
  window(Indent,9,80-Indent-1,19);
  ClrScr;
  writeln('   1. Calibrate SLA-1 Machine');
  writeln;
  writeln('   2. Process Calibration File');
  writeln;
  writeln('   3. Install New Geometric Table');
  writeln;
```

```
  writeln('   4. Explanation of Calibration Process');
  writeln;
  writeln('   5. Initialize Digital Potentiometer');
  writeln;
  write  ('       ( X to Exit )');
  window(1,1,80,25);
  gotoxy(1,22);
  centerln('Please make a selection.');
end;

procedure reboot;
begin
   port[$305]:= port[$305] or 7;

Inline($B8/$40/$00/$8E/$D8/$BB/$72/$00/$C7/$07/$34/$00/$
C7/$47/$01/$12/$00/
          $EA/$00/$00/$FF/$FF/
          $01/$04/$43/$4F/$50/$59/$03/$61/$26/$05/$50/
$41);
end;

procedure UserWait;
begin
   writeln;
   writeln;
   writeln('       ... press any key to continue ');
   repeat delay(100) until KeyPressed;
   junkkey:=Readkey;
end;

procedure InstallNewGeometricCorrectionFile;
const
  AnyFile = $3F;
var
  i : integer;
  DirInfo : SearchRec;
begin
```

```
   ClrScr;
   gotoxy(1,13);
   writeln('   Warning:  Installing the geometric
correction table will delete');
   writeln('      the existing \3DSYS\STEREO.GEO file by
copying the');
   writeln('      selected \CAL\*.GEO file over it.  ');
   writeln;
   writeln('      Continue ? ');
   repeat delay(100) until keypressed;
   key := upcase(readkey);
   if key <> 'Y' then exit;
   ClrScr;
   writeln;
   writeln;
   writeln('   SLA-1 Calibration Control Program -
Install Geometric Table');
   writeln;
{   exec('c:\command.com','/c dir \CAL\*.GEO'); }
   writeln('   Existing Geometric Correction Files: ');
   write  ('          (none)');
   gotoxy(WhereX-16,WhereY);
   i := 0;
   FindFirst('\CAL\*.GEO',anyfile,DirInfo);
   while DosError = 0 do begin
      if (i - 4) = 0 then begin
         writeln;
         i := 0;
      end;
      gotoxy(8+(15*i),WhereY);
      i := i + 1;
      write(DirInfo.Name);
      FindNext(DirInfo);
   end;
   writeln;
   gotoxy(1,WhereY);
   writeln;
```

```
   CursorOn;
   FileOk := false;
   repeat
      writeln('   Please enter the file to install or X
to exit.');
      readln(InputStr);
      for i := 1 to length(InputStr) do
InputStr[i]:=upcase(InputStr[i]);
      if InputStr[1]='X' then begin
         CursorOff;
         exit;
      end;
      if pos('.',InputStr)<>0 then InputStr := '\CAL\' +
InputStr
                                 else InputStr := '\CAL\' +
InputStr + '.GEO';
      FindFirst(InputStr,AnyFile,DirInfo);
      writeln;
      if DosError = 0 then FileOK := true
      else writeln('   Error:  Invalid file name
input.');
   until FileOK;
   exec('c:\command.com','/c copy'+InputStr+'
\3DSYS\STEREO.GEO');
   delay(500);
   if DOSerror = 0 then begin
      writeln;
      writeln('   The system will now automatically
reboot.');
      UserWait;
      reboot;
   end else begin
      CursorOff;
      writeln('   Error: Invalid file name input.');
      UserWait;
   end;
end;
```

```
procedure Explain;
begin
   ClrScr;
   writeln;
   writeln('   The calibration process provides data to
translate the absolute');
   writeln('   laser movement coordinates to the actual
locations in the SLA-1');
   writeln('   vat. ');
   writeln;
   writeln('   The first step in calibrating utilizes a
plate with many sensing');
   writeln('   holes which the beam moves across.   The
absolute coordinates for');
   writeln('   each sensing hole are stored in a *.CAL
file.  Many calibration');
   writeln('   files can be generated.  They are stored
in the \CAL directory');
   writeln('   as *.CAL where the * is a time stamp.  A
calibration file run on');
   writeln('   May 1 at 1:06 PM would be named
05011306.CAL (modytime.CAL).');
   writeln;
   writeln('   The *.CAL file does not compensate for
hardware mirror drift.');
   writeln('   Between each sensor hole reading on the
calibration plate the ');
   writeln('   laser recalculates the (static) sensor 1
and 2 positions.  B  );
   writeln('   comparing the drift of sensors 1 and 2
the drift of the entire');
   writeln('   mirror apparatus can be computed.  These
results are in contained');
   writeln('   in the *.DRF file.');
   writeln;
```

```
   writeln('   The geometric (correction) table is the
culmination of the ');
   writeln('   translation data.  This table only
becomes active when the *.GEO');
   writeln('   file from the \CAL directory is moved to
the \3DSYS directory');
   writeln('   as STEREO.GEO and the system is rebooted.
(press any to exit)');
   repeat delay(100) until keypressed;
   key := Readkey;
end;

procedure InitDigitalPot;
var
   ADC : real;
   i : integer;
   password1 : string;
   WiperSettings : array[1..100] of real;

procedure ShowWiperMenu1;
begin
   ClrScr;
   gotoxy(1,8);
   writeln(' Digitally controlled potentiometer settings
should only be ');
   writeln(' executed by 3D systems field engineers. ');
   writeln;
   writeln;
   write  (' Please key password:');
end;

procedure ShowWiperMenu2(ADC : real;  Status : char);
var
   StatusStr : string;
begin
  if Status = 'U' then StatusStr := 'UP    '
   else if Status = 'D' then StatusStr := 'DOWN  '
```

```
   else if Status = 'S' then StatusStr := 'STABLE';
   ClrScr;
   gotoxy(1,8);
   writeln(' Voltage Wiper Element Status =',StatusSTR);
   writeln;
   writeln(' Current sensor ADC x100 =',ADC:4:6);
   writeln;
   writeln;
   writeln('    ( Initalization is complete only when
status = STABLE ) ');
   delay(500);
end;

procedure MoveWiper(direction:char;times:integer);
var
   iwaste1,iwaste2,i : integer;
begin
   { port[$305] bit 4 = CS, 5 = U/D, 6 = INC    }
   { MSB 76543210 LSB  ===> CS is bit 4 = 00010000 =
$10 }
   { NMI is bit 0-2 = $07 }
   port[$305] := port[$305] or $F0;           { set all
hi initially)
   for iwaste1 := 1 to 10 do inc(iwaste2);
   port[$305] := port[$305] and $EF;          { set CS
lo to set-up          }
   for iwaste1 := 1 to 1 do inc(iwaste2);   { 0 delay
for CS to INC       }
         for iwaste1 := 1 to 1 do inc(iwaste2);{ 100 n }
         if direction = 'U'
         then port[$305] := port[$305] or  $20  { set
for upward increment  }
         else port[$305] := port[$305] and $DF; { set
for downward increment)
         for iwaste1 := 1 to 1 do inc(iwaste2); { 4 us
AC timing delay      }
      for i := 1 to times do begin
```

```
        port[$305] := port[$305] and $BF;      { use
INC to move wiper       }
        for iwaste1 := 1 to 10 do inc(iwaste2);{ 50 u
delay before voltage x }
                                                { this
covers the 2u INC to CS}
        port[$305] := port[$305] or $40;
{ set incl hi }
        for iwaste1 := 1 to 1 do inc(iwaste2);
     end;
  port[$305] := port[$305] or $10;              { set
cs hi to save setup}
  delay(50);                                    { 50 ms
delay for CS deselect }
end;

procedure DecWiper100times;
var
  i : integer;
begin
  ShowWiperMenu2(0,'D');
  MoveWiper('D',100);
end;

procedure IncWiper100times;
var
  i,j: integer;
  sum: real;
begin
  sum := 0;
  for j:= 1 to 10 do begin
     ADC := ReadADC(SensorChannel[Sensor]+10);
     sum := sum + (200 - ADC);
     ADC := sum / 10;
  end;
  WiperSettings[1] := ADC;
  i := 2;
```

```
   repeat
      inc(i);
      MoveWiper('U',1);
      sum := 0;
      for j:= 1 to 10 do begin
         ADC := ReadADC(SensorChannel[Sensor]+10);
         sum := sum + (200 - ADC);
         ADC := sum / 10;
      end;
      WiperSettings[i] := ADC;
      ShowWiperMenu2(ADC,'U');
   until (ADC <= 0) or (i = 100);
   MoveWiper('D',1);
   if i = 100 then i := 99;
   WiperSettings[i+1] := 999;
end;

procedure OutputValues;
var
  junkkey : char;
  i : integer;
begin
   ShowWiperMenu2(ADC,'S');
   repeat delay(100) until keypressed;
   junkkey := ReadKey;
   ClrScr;
   for i := 1 to 25 do begin
      write  ('    W[',i   ,']= ',WiperSettings[i
]:3:3);
      wr  e  ('    W[',i+25,']=
',Wipers  tings[i+25]:3:3);
      wr_ e  ('    W[',i+50,']=
',WiperSettings[i+50]:3:3);
      write  ('    W[',i+75,']=
',WiperSettings[i+75]:3:3);
      if i <> 25 then writeln;
   end;
```

```
   repeat delay(100) until keypressed;
   junkkey := ReadKey;
end;

begin
   ShowWiperMenu1;
   readln(password1);
   if password1 = '3dsys' then begin
         Sensor := 2;
         for i := 1 to 100 do WiperSettings[i] := 0;
         DecWiper100times; { start our search for the
lowest ADC at bottom }
         IncWiper100times; { and save the ADC reading
for each one }
         OutputValues;     { this is temporary for print
screen }
   end;
end; { procedure InitDigitalPot }

begin
  CursorOff;
  indent := 18;
  repeat
    ShowCaliMenu;
    GetSelection;
    ClrEol;
    case key of
      '1': Calibr8;
      '2': begin
             exec('c:\command.com','/c CALOUT');
             delay(3000);
           end;
      '3': InstallNewGeometricCorrectionFile;
      '4': begin
             Explain;
             key := 'z';
           end;
```

```
            '5': InitDigitalPot;
            'X': begin
                    key := 'z';
                    Exit;
                 end;
          end;
          key := 'z';
        until false;
      end;
```

```
{-------------------------------------------------------

                BUILD Beam Monitoring Routines

-------------------------------------------------------}

{
   Hardware I/O
}
function ReadLaserBoard(Channel: Integer): Integer;
var
  count: LongInt;
  ADC,i: Integer;
begin
  count := Timeout;
  port[LaserAddr+12] := Channel;
  { must wait 10 us for EOS to go LO (286 @ 4us/loop) }
  i := 1;
  while i < 10 do inc(i);
  repeat   { then approx 110 us for conversion with
signal D7 HI }
    dec(count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if PeakReading < ADC then PeakReading := ADC;
  if count = 0 then ReadLaserBoard := -1
    else ReadLaserBoard := ADC;
end;

procedure ReadTemperature(var Resist,Temper: FloatType);
var
  temp: FloatType;
  It,ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board
```

```
  use 1/T, LOG(R) function TO linearize the
interpolation
}
  ADC := ReadLaserBoard(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC);    { from
20K resistor on JB Board }
  if ADC>214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] > Resist)
do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It]));
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) - ln(Resist)));
  end;
  Temper := Temper - 273.15;
end;

procedure println(text: String);
begin
  if not transparent then writeln(text);
end;


{-------------------------------------------------------

                  BEAM SCANNING ROUTINES

/-------------------------------------------------------
```

```
                                        ************
                                        * MoveBeam *
                                        ************

  Routine to move mirrors to X,Y directly
}
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 then pos := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
 { x is bit 0, y is bit 1 }
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInXAxis(x: LongInt);
var
  stepX: LongInt;
begin
  if abs(x - MonOldX) > SlewLimitMaxStep then begin
    if x - MonOldX > 0 then begin
      stepX := MonOldX + SlewLimitMaxStep;
      while stepX < x do begin
        AxisMove(stepX,1);
        inc(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepX := MonOldX - SlewLimitMaxStep;
      while stepX > x do begin
        AxisMove(stepX,1);
        dec(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end;
  end;
```

```
  AxisMove(x,1);
  MonOldX := x;
end;

procedure MoveInYAxis(y: LongInt);
var
  stepY: LongInt;
begin
  if abs(y - Mon  dY) > SlewLimitMaxStep then begin
    if y - MonOl  > 0 then begin
      stepY := MonOldY + SlewLimitMaxStep;
      while stepY < y do begin
        AxisMove(stepY,2);
        inc(stepY,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepY := MonOldY - SlewLimitMaxStep;
      while stepY > y do begin
        AxisMove(stepY,2);
        dec(stepY,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end;
  end;
  AxisMove(y,2);
  MonOldY := y;
end;

procedure MoveBeam(x,y: LongInt);
begin
  MoveInXAxis(x);
  MoveInYAxis(y);
end;
```

```
{
                                    ******************
                                    * GetBeamProfile *
                                    ******************

}

procedure SetupBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer;X,Y:
FloatType;_ScanStep,_BkgndThreshold: Integer);
begin
  with BeamProfileVars[Sensor] do begin
    if X                >= 0 then Xpos           :=
LibTrunc(X);
    if Y                >= 0 then Ypos           :=
LibTrunc(Y);
    if _ScanStep        >= 0 then ScanStep       :=
_ScanStep;
    if _BkgndThreshold >= 0 then BkgndThreshold :=
_BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;

procedure GetBeamProfile(var BeamProfileVars:
BeamProfileVarType;
  Sensor: Integer);
var
  Pass,row,col,ADC,delay,waste: Integer;
  Xcorner,Ycorner: LongInt;
begin

  with BeamProfileVars[Sensor] do begin
    XYweight        := 0;
```

```
    Xweight          := 0;
    Yweight          := 0;
    TotalWeight      := 0;    { for power, not removing
noise till end }
    SaturationCount := 0;
    HiPoints         := 0;
    BeamFound        := false;
    for col:=1 to ColMax do ColWeight[col] := 0;
    for row:=1 to RowMax do RowWeight[row] := 0;
    Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
    Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
    MoveBeam(Xcorner,Ycorner);
    waste := 0;
    while waste < InitialDelay do inc(waste);
    PeakReading := 0;
    for col:=1 to ColMax do begin    { prefer to return
around edges after line }
      Xpos := Xcorner + col * ScanStep;
      Ypos := Ycorner;
      MoveBeam(Xpos,Ypos);
      waste := 0;
      while waste < ColToColDelay do inc(waste);
      for row:=1 to RowMax do begin
        inc(Ypos,ScanStep);
        MoveInYAxis(Ypos);
        waste := 0;
        while waste < RowToRowDelay do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC < BkgndThreshold then ADC := 0
          else begin
            inc(HiPoints);
            BeamFound := true;
          end;
        if ADC > 210 then inc(SaturationCount);
        inc(TotalWeight,ADC);
        inc(ColWeight[col],ADC);
        inc(RowWeight[row],ADC);
```

```
          BeamMap[row,col] := ADC;
          inc(XYweight,ADC);
          inc(Xweight,Xpos * ADC);
          inc(Yweight,Ypos * ADC);
        end; {for row}
      end; {for col}
      MaxSensorReading := PeakReading;
    end;
  end;

  {
                                    ************
                                    * GoodRead *
                                    ************
  }

  procedure GoodRead(Sensor,iter: Integer);
  var
    Try: Integer;
  begin
    with BeamProfileVariables[Sensor] do begin
      Xaverage  := 0;
      Yaverage  := 0;
      TotWtAvg  := 0;
      GoodCount := 0;
      for Try := 1 to iter do begin
        SetupBeamProfile(BeamProfileVariables,Sensor,Xsp
  [Sensor],
          Ysp[Sensor],-1{ScanStep},-1{BkgndThreshold});
        GetBeamProfile(BeamProfileVariables,Sensor);
        if XYweight > 0 then begin
          Xaverage := Xweight / XYweight + Xaverage;
          Yaverage := Yweight / XYweight + Yaverage;
          TotWtAvg := TotWtAvg + TotalWeight -
  SensorBkgnd[Sensor];
          inc(GoodCount);
```

```
      end;
      if keypressed then HandleKeyboardRequest;
    end; {for Try}
    if GoodCount > 0 then begin
      Xaverage := Xaverage / GoodCount;
      Yaverage := Yaverage / GoodCount;
      TotWtAvg := TotWtAvg / GoodCount;
    end;
    Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
  end;
end;

procedure IOSetup;
begin
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
end;

{
                                    ************
                                    * FindBeam *
                                    ************
}

procedure SetupFindBeam(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer;X,Y:
FloatType;_SpiralSideMax,_SpiralTries,_ScanStep,
  _BkgndThreshold,_SpiralDelay,_NumCenterPasses:
Integer);
begin
  with FindBeamVars[Sensor] do begin
    if X                >= 0 then Xval          :=
LibTrunc(X);
```

```
    if Y                >= 0 then Yval            :=
LibTrunc(Y);
    if _SpiralSideMax   >= 0 then SpiralSideMax   :=
_SpiralSideMax;
    if _SpiralTries     >= 0 then SpiralTries     :=
_SpiralTries;
    if _ScanStep        >= 0 then ScanStep        :=
_ScanStep;
    if _SpiralDelay     >= 0 then SpiralDelay     :=
_SpiralDelay;
    if _BkgndThreshold  >= 0 then BkgndThreshold  :=
_BkgndThreshold;
    if _NumCenterPasses >= 0 then NumCenterPasses :=
_NumCenterPasses;
  end;
end;

function SpiralSearchForHole(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer): Boolean;
var
  i,ADC,SpiralSide,waste: Integer;
  X,Y: LongInt;
begin
  with FindBeamVars[Sensor] do begin
    X := LibTrunc(Xval);
    Y := LibTrunc(Yval);
    MoveBeam(X,Y);
    delay(SpiralDelay);
    BeamFound := false;
    SpiralSide := 1;
    while (SpiralSide <= SpiralSideMax) and not
BeamFound do begin
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInXAxis(X);
        waste := 0;
```

```
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(X,ScanStep);
        inc(i);
        if keypressed then HandleKeyboardRequest;
      end;
      i := 1;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInYAxis(Y);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then BeamFound := true
        else inc(Y,ScanStep);
        inc(i);
        if keypressed then HandleKeyboardRequest;
      end;
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := BeamFound;
  end;
end;

function FindBeam(var FindBeamVars:
FindBeamVarType;Sensor: Integer;
  FindMethod: FindMethodType): Boolean;
const
  HoleSeparation = 600;
var
  i,ADC,pass,CantFind,SpiralSide: Integer;
```

```
  XspAv,YspAv: FloatType;
  X,Y,GoodX,GoodY: LongInt;
  BeamFound: Boolean;
begin
  FindBeam := true;
  with FindBeamVars[Sensor] do begin
    CantFind  := 0;
    repeat
      X := LibTrunc(Xval);
      Y := LibTrunc(Yval);
      MoveBeam(X,Y);   { gets us to best starting
position }
{
  now spiral to search for holes
}
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
        -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      BeamFound := SpiralSearchForHole
(FindBeamVariables,Sensor);
      if not BeamFound then begin
        SetupFindBeam(FindBeamVariables,Sensor,X,Y,
SpiralSideMax * 2,
          -1{SpiralTries},-1{ScanStep},BkgndThreshold -
1,-1{SpiralDelay},
          -1{NumCenterPasses});
        inc(CantFind);
      end;
    until (CantFind >= SpiralTries) or BeamFound;
    if CantFind >= SpiralTries then begin
      MoveBeam(Xval,Yval);
      FindBeam := false;
      exit;
    end;
    X := Xval;
```

```
    Y := Yval;
{
  Verify that we have 2 mil hole...

  1/2 mil    1 mil ) left rear              2 mil     4
mil ) right front
                    ) of vat
  ) of vat
   4  mil    2 mil )                        1 mil    1/2
mil )

     Sensor #1                                  Sensor #2
```

We have the position of one of the four holes: 1/2, 1, 2, or 4 mils. We want to find the 2 mil hole closest to the vat. Note that the orientation of the holes is different for Sensors 1 and 2.

Hole Size Check Procedure:

1. We may have the position for the 1/2 or 1 mil hole. Move towards the vat in the X-axis and search for the 1 or 2 mil hole. If the beam is seen, adopt the new coordinates.

2. Now we may have the position for the 4 mil hole. Move towards the vat in the Y-axis and search for the 2 mil hole. If the beam is seen, use the new coordinates.

```
}
    if (Sensor < 3) and (FindMethod = Verify2MilHole)
then begin
      GoodX := X;
      GoodY := Y;
      if X < 32767 then inc(X,HoleSeparation)
        else dec(X,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,
SpiralSideMax div 2,
```

```
        -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
          GoodX := Xval;
          GoodY := Yval;
        end;
      X := GoodX;
      Y := GoodY;
      if Y < 32767 then inc(Y,HoleSeparation)
        else dec(Y,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
        -1{SpiralTries},-1{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(FindBeamVariables,Sensor)
then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
      X := GoodX;
      Y := GoodY;
    end;

    XspAv := 0;
    YspAv := 0;
    if BeamFound then begin
      for pass:=1 to NumCenterPasses do with
BeamProfileVariables[Sensor] do begin
        CantFind := 0;
        { discard the first }

SetupBeamProfile(BeamProfileVariables,Sensor,X,Y,
          4{ScanStep},LibTrunc(Bk[Sensor] + 0.5));
        repeat
```

```
          GetBeamProfile(BeamProfileVariables,Sensor);
          if XYweight = 0 then inc(CantFind);
        until (XYweight > 0) or (CantFind > 3);
        if CantFind > 3 then begin
          FindBeam := false;
          exit;
        end;
{
  Later select sensor hole size automatically
}
        X := LibTrunc(Xweight / XYweight);   { better
coordinates }
        Y := LibTrunc(Yweight / XYweight);
        Power[Sensor] := (TotalWeight -
SensorBkgnd[Sensor])
          * (SensorCalib[Sensor,3]);
        XspAv := XspAv + Xweight / XYweight;
        YspAv := YspAv + Yweight / XYweight;
      end; {for pass}
      if NumCenterPasses > 1 then begin
        XspAv := XspAv / NumCenterPasses;
        YspAv := YspAv / NumCenterPasses;
      end;
    end; {if BeamFound}
    Xval := X;       { update integer coordinates }
    Yval := Y;
    Xavg := XspAv;   { update average coordinates }
    Yavg := YspAv;
  end;
end;

function CoarseFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
```

```
    LibTrunc(Bk[Sensor] +
0.5)(BkgndThreshold),1(SpiralDelay),
    1(NumCenterPasses));
  if FindBeam(FindBeamVariables,Sensor,NormalFind) then
begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  CoarseFindBeam := BeamFound;
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    50(SpiralSideMax),3(SpiralTries),16(ScanStep),
    LibTrunc(Bk[Sensor] + 0.5)(BkgndThreshold),100
(SpiralDelay),
    3(NumCenterPasses));
  if FindBeam(FindBeamVariables,Sensor,Verify2MilHole)
then begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  FineFindBeam := BeamFound;
end;

function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) then
    BeamFound := FineFindBeam(Sensor);
  FindLaserBeam := BeamFound;
end;
```

(- - - - -)

```
function FindHoles2(Sensor: Integer): Boolean;
{
  When we want to use new starting point
   updates hole position
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;

function FindHoles(Sensor: Integer): Boolean;
var
  BkRead: LongInt;
  ADC,row,col: Integer;
begin
  Xpos := XRest;
  Ypos := YRest;
(){  MoveBeam(Xpos,Ypos); }
  if Sensor = 1
  then MoveBeam(LibTrunc(Xsp[Sensor]),LibTrunc(Ysp
[Sensor]-2000));
  if Sensor = 2
  then MoveBeam(LibTrunc(Xsp[Sensor]),LibTrunc(Ysp
[Sensor]+2000));
  BkRead := 0;
  for col:=1 to ColMax do begin
    for row:=1 to RowMax do begin
      ADC := ReadLaserBoard(SensorChannel[Sensor]);
      inc(BkRead,ADC);
    end;
  end;
  if BkRead>65535 then BkRead:=65535;
  SensorBkgnd[Sensor] := BkRead;
  { remove noise for centering }
  Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax) + 3;
  FindHoles := FindHoles2(Sensor);
end;
```

```
{=========================================================}

{
  Read BEAM.PRM parameters file
}
procedure ReadBeamParamsFile;
var
  ParamFile: text;
  i,count,code,code1,code2,Sensor,Pinhole: Integer;
  cmd,ParamLine: String;

  procedure GetParamFileLine;
  begin
    if eof(ParamFile) then ParamLine := ''
    else repeat
      readln(ParamFile,ParamLine);
    until eof(ParamFile) or ((ParamLine <> '') and
(ParamLine[1] <> '!'));
  end;

begin
  {$I-}
  assign(ParamFile,BeamParamFileName);
  reset(ParamFile);
  if IOresult<>0 then begin
    assign(ParamFile,HomeDirectory+BeamParamFileName);
    reset(ParamFile);
    if IOresult<>0 then halt(10);
  end;
  {$I+}
  GetParamFileLine;
  if copy(ParamLine,1,1) = chr(34) then begin    { handle
quoted options line }
    i := pos(chr(34),copy(ParamLine,2,255));
    if i>0 then ParamLine[0] := chr(i);
```

```
    ParamLine := NoQuotes(ParamLine);
    GetParamFileLine;
  end;
  XRest := WordVal(ParamLine,code);
  YRest := WordVal(copy(ParamLine,code+1,255),code);
  GetParamFileLine;
  SensorNum := WordVal(ParamLine,code);
  for Sensor:=1 to SensorNum do begin
    GetParamFileLine;
    SensorChannel[Sensor] := WordVal(ParamLine,code);
    GetParamFileLine;
    XSensorPos[Sensor] := WordVal(ParamLine,code);
    YSensorPos[Sensor] :=
WordVal(copy(ParamLine,code+1,255),code);
    for Pinhole:=1 to 4 do begin
      GetParamFileLine;
      SensorCalib[Sensor,Pinhole] :=
RealVal(ParamLine,code);
    end;
  end; {for Sensor}
  GetParamFileLine;
  TempChannel := WordVal(ParamLine,code);   { assume
only one thermometer at first }
  GetParamFileLine;
  TempCalPts := WordVal(ParamLine,code);
  for i:=1 to TempCalPts do begin
    { Resistance[i],Temperature[i] may not be integers }
    GetParamFileLine;
    Resistance [i] := RealVal(ParamLine,code);
    Temperature[i] :=
Re  Val(copy(ParamLine,code+1,255),code) + 273.15;
  end;
  for i:=1 to 3 do begin   { read in Fast, Medium, &
Slow Calib Delay values }
    GetParamFileLine;
    CalibDelays[i,1] := LongVal(ParamLine,code1);
```

```
    CalibDelays[i,2] :=
LibTrunc(RealVal(copy(ParamLine,code1+1,255),code2)*10+0
.5);
    CalibDelays[i,3] := LibTrunc(RealVal(copy(ParamLine,
code2+code1+1,255),code)*10+0.5);
  end;
  SlewLimitMaxStep := 1000;
  SlewLimitDelay   := 2;
  if not eof(ParamFile) then begin
    GetParamFileLine;
    SlewLimitMaxStep := WordVal(ParamLine,code);
    if SlewLimitMaxStep < 10 then SlewLimitMaxStep :=
10;
    GetParamFileLine;
    SlewLimitDelay   := WordVal(ParamLine,code);
  end;
  close(ParamFile);
  Resistance [0] := Resistance[1]  * 2 - Resistance [2];
  Temperature[0] := Temperature[1] * 2 - Temperature[2];
end;


{-----------------------------------------------------}

procedure SetupMonitoring;
begin
  if not noDriftFlag then begin
     port[LaserAddr+3] := $88;
     for i:=1 to 2 do begin
       Xsp[i] := XSensorPos[i];
       Ysp[i] := YSensorPos[i];
     end;
  end;
end;

procedure CantLocateSensor(num: Integer);
begin
```

```
  PrintError('Where''s Profiler Sensor #' + IntStr(num)
    + ' on layer ' + IntStr(ControlLayer) + '?');
end;

procedure MonitorLaserBeamDrift;
type
  BufType = array[1..19] of Byte;
const
  XYBufIndex: array[2..5] of Integer = (2,6,11,15);
  CmdBufMask: BufType = (STEREO_D1,0,0,0,0,0,0,0,0,
                         STEREO_D2,0,0,0,0,0,0,0,0,0);
var
  i: Integer;
  CmdBuf: BufType;
  GoodProfile: Boolean;
begin

  if not (NoDriftFlag or LaserOff) then begin
    if StatFlag then
BenchStart(ord(BENCH_BUILD_ANALYZE));
    Action(ACTION_ANALYZING);
    CmdBuf := CmdBufMask;
    GoodProfile := true;
 (*   MonOldX := XRest;    { BEAM.PRM rest should be
close to BUILD.PRM rest pos }
    MonOldY := YRest;
    AxisMove(MonOldX,1);
    AxisMove(MonOldY,2);
    delay(250); *)
    Sensor := 1;
    while (Sensor <= 2) and GoodProfile do begin
    if Sensor <> SAVESENS then begin
    {){ Beep; }
       MoveBeam(0,65500);
       SAVESENS := Sensor;
    end;
```

```
      MoveBeam(LibTrunc(Xsp[Sensor]),LibTrunc(Ysp
[Sensor]));
      delay(250);
      if FindHoles(Sensor) then begin
        with FindBeamVariables[Sensor] do begin
          Xsp[Sensor] := Xavg;   { for fast locate next
time }
          Ysp[Sensor] := Yavg;
          {$IFDEF COP}
          Move(Xavg,CmdBuf[XYBufIndex[Sensor * 2]],4);
          Move(Yavg,CmdBuf[XYBufIndex[Sensor * 2 +
1]],4);
          {$ELSE}
          ConvertRtoS(@Xavg,@CmdBuf[XYBufIndex[Sensor *
2]]);
          ConvertRtoS(@Yavg,@CmdBuf[XYBufIndex[Sensor *
2 + 1]]);
          {$ENDIF}
        end;
      end else begin
        CantLocateSensor(Sensor);
        GoodProfile := false;
      end;
      inc(Sensor);
    end;
    if GoodProfile then
ShowError(BinarySTEREO(@CmdBuf));
    LaserExecute;   { force beam back to BG rest
position }
    if StatFlag then
BenchStop(ord(BENCH_BUILD_ANALYZE));
  end;
end;
```

```
{

   Options
   Build Options Pop-Up Windows Include File
   3D Systems Stereolithography System

}

const
  LeftSide  = 5;
  RightSide = 219;
  MaxOptionItem = 23;

var
  BoxTop,BoxBtm: Integer;
  ScreenOverlay,InputOverlay: Pointer;
  SolidBlock: array[1..1300] of Word;

procedure InitOptionsMenu;
begin
  GetMem(ScreenOverlay,32000);
  GetMem(InputOverlay,3000);
  OptionItem := 1;
  SolidBlock[1] := RightSide-LeftSide-3;
  SolidBlock[2] := 10;
  for i:=3 to 1300 do SolidBlock[i] := $FFFF;
end;

procedure HighlightOption;
begin
  PutImage(LeftSide+1,(OptionItem+2)*10-2,SolidBlock,
XorPut);
end;

function OnOff(Flag: Boolean): String;
begin
  if Flag then OnOff := 'On'
```

```
    else OnOff := 'Off';
end;

function PlusMinus(Flag: Boolean): Char;
begin
  if Flag then PlusMinus := '-'
    else PlusMinus := '+';
end;

procedure UpdateItemLineData(Text: str16;Spacing:
Integer);
begin
  HighlightOption;   { unhighlight actually }
  MoveTo(122,(OptionItem+2)*10);
  OverText(Text,Spacing);
end;

procedure ToggleOnOff(var Flag: Boolean);
begin
  Flag := not Flag;
  if Flag then UpdateItemLineData('On',3)
    else UpdateItemLineData('Off',3);
end;

procedure TogglePlusMinus(var Flag: Boolean);
begin
  Flag := not Flag;
  if Flag then UpdateitemLineData('-',1)
    else UpdateItemLineData('+',1);
end;

procedure SetupInputBox;
begin
  BoxTop := (OptionItem+3)*10-2;
  BoxBtm := (OptionItem+6)*10-2;
  GetImage(100,BoxTop,300,BoxBtm,InputOverlay^);
  SetViewPort(160,60+BoxTop,360,60+BoxBtm,true);
```

```
  ClearViewPort;
  Rectangle(0,0,360-160-1,BoxBtm-BoxTop-1);
  Rectangle(4,4,360-160-5,BoxBtm-BoxTop-5);
  MoveTo(10,11);
end;

procedure EraseInputBox;
begin
  SetViewPort(60,60,280,330,true);
  PutImage(100,BoxTop,InputOverlay^,NormalPut);
end;

procedure IntInput(var value: LongInt);
begin
  SetupInputBox;
  value :=
LongVal(GetInput(16,NoSpecial,LongStr(value)),code);
  EraseInputBox;
  UpdateItemLineData(LongStr(value),12);
end;

procedure NonZeroActiveInput(var value: LongInt);
begin
  SetupInputBox;
  value :=
LongVal(GetInput(16,NoSpecial,LongStr(value)),code);
  EraseInputBox;
  if value <> 0 then
UpdateItemLineData(LongStr(value),10)
    else UpdateItemLineData('Off',12);
end;

procedure RealInput(var value: FloatType);
begin
  SetupInputBox;
  value := RealVal(copy(GetInput(16,NoSpecial,RealStr
(value,4)),1,10),code);
```

```
  EraseInputBox;
  UpdateItemLineData(RealStr(value,4),12);
end;

procedure EditOption;
begin
  case OptionItem of
     1: IntInput(StartZ);
     2: IntInput(EndZ);
     3: IntInput(PauseAfterLayer);
     4: ToggleOnOff(SwapXY);
     5: TogglePlusMinus(NegX);
     6: TogglePlusMinus(NegY);
     7: ToggleOnOff(LaserOff);
     8: ToggleOnOff(ZTableOff);
    12: ToggleOnOff(NegZStage);
    13: ToggleOnOff(LayerCtrl);
    14: ToggleOnOff(NoDriftFlag);
    15: ToggleOnOff(SweepFlag);
    19: NonZeroActiveInput(DemoOutOfVatDelay);
    20: RealInput(PartHeight);
    21: ToggleOnOff(TextFlag);
    22: ToggleOnOff(StatFlag);
    23: ToggleOnOff(ShowStuff);
  end;
end;

procedure IncOptionItem;
begin
  inc(OptionItem);
  if OptionItem in [9,16] then inc(OptionItem,3);
end;

procedure DecOptionItem;
begin
  dec(OptionItem);
  if OptionItem in [11,18] then dec(OptionItem,3);
```

```
end;

function ValidCycleKey: Boolean;
begin
  ValidCycleKey := false;
  if   (ordkey = CrKey)       or (ordkey = EscKey)
    or (ordkey = UpArrowKey) or (ordkey = DownArrowKey)
    or (ordkey = HomeKey)     or (ordkey = EndKey)
    then ValidCycleKey := true;
end;

procedure CycleSelectOptions;
begin
  repeat
    HighlightOption;
    repeat
      repeat until keypressed;
      GetBuildKey;
      if not ValidCycleKey then beep;
    until ValidCycleKey;
    if key = Cr then EditOption else begin
      HighlightOption;   { unhighlight }
      case ordkey of
        EscKey      : ;
        UpArrowKey  : if OptionItem = 1 then OptionItem
:= MaxOptionItem
                        else DecOptionItem;
        DownArrowKey: if OptionItem = MaxOptionItem then
OptionItem := 1
                        else IncOptionItem;
        HomeKey     : OptionItem := 1;
        EndKey      : OptionItem := MaxOptionItem;
      end;
    end;
  until key = Esc;
end;
```

```
procedure OptionsMenu;
var
  ss: LongInt;
begin
  SetActivePage(VisualPageNumber);
  GetImage(60,60,285,330,ScreenOverlay^);
  SetViewPort(60,60,285,330,true);
  ClearViewPort;
  SetTextJustify(LeftText,TopText);
  SetTextStyle(DefaultFont,HorizDir,1);
  if GraphMode=EGAMonoHi then SetColor(White)
{ prevent blinking }
    else SetColor(LightMagenta);
  Rectangle(0,0,224,269);
  Rectangle(4,4,219,265);
  OutTextXY(46, 10,'Standard Options');
  Line(LeftSide,23,RightSide,23);
  OutTextXY(42, 30,'/START    '+LongStr(StartZ));
  OutTextXY(42, 40,'/STOP     '+LongStr(EndZ));
  OutTextXY(42, 50,'/PAUSE
'+LongStr(PauseAfterLayer));
  OutTextXY(42, 60,'/SWAPON   '+OnOff(SwapXY));
  OutTextXY(42, 70,'/X        '+PlusMinus(NegX));
  OutTextXY(42, 80,'/Y        '+PlusMinus(NegY));
  OutTextXY(42, 90,'/LOFF     '+OnOff(LaserOff));
  OutTextXY(42,100,'/ZOFF     '+OnOff(ZTableOff));
  Line(LeftSide,113,RightSide,113);
  OutTextXY(26,120,'Compatibility Options');
  Line(LeftSide,133,RightSide,133);
  OutTextXY(42,140,'/NEGZ     '+OnOff(NegZStage));
  OutTextXY(42,150,'/LC       '+OnOff(LayerCtrl));
  OutTextXY(42,160,'/NODRIFT  '+OnOff(NoDriftFlag));
  OutTextXY(42,170,'/SWEEP    '+OnOff(SweepFlag));
  Line(LeftSide,183,RightSide,183);
  OutTextXY(30,190,'Demo & Debug Options');
  Line(LeftSide,203,RightSide,203);
  if DemoOutOfVatDelay<>0
```

```
    then OutTextXY(42,210,'/DEMO
'+LongStr(DemoOutOfVatDelay))
      else OutTextXY(42,210,'/DEMO     Off');
  OutTextXY(42,220,'/HEIGHT
'+copy(RealStr(PartHeight,4),1,12));
  OutTextXY(42,230,'/TEXT     '+OnOff(TextFlag));
  OutTextXY(42,240,'/STAT     '+OnOff(StatFlag));
  OutTextXY(42,250,'/SHOW     '+OnOff(ShowStuff));
  CycleSelectOptions;
  SetViewPort(0,0,GetMaxX,GetMaxY,true);
  PutImage(60,60,ScreenOverlay^,NormalPut);
  SetActivePage(OnScreen);
  SetColor(Cyan);
end;
```

```
procedure BenchInit;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    for i:=1 to BENCH_ITEMS do begin
      Bench_Item_Total[i]:=0;
      Bench_Item_Stamp[i]:=0;
    end;
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    writeln(BenchFile,'Benchmark Started for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
    Bench_Start_DOW := DOW;
  end;


function GetS100Time : LongInt;
  var
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    if DOW<Bench_Start_DOW then Inc(DOW,7);
    GetS100Time := S100 + LongInt(Sec)*100 +
LongInt(Min)*6000 + Hr*360000 +
                  (DOW-Bench_Start_DOW)*8640000;
  end;
```

```
procedure BenchStart(Item : Word);
  begin
    if Bench_Item_Stamp[Item]=0 then
Bench_Item_Stamp[Item]:=GetS100Time;
  end;

procedure BenchStop(Item : Word);
  begin
    if Bench_Item_Stamp[Item]>0 then begin
      Inc(Bench_Item_Total[Item],GetS100Time -
Bench_Item_Stamp[Item]);
      Bench_Item_Stamp[Item]:=0;
    end;
  end;

procedure BenchEnd;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    for i:=1 to BENCH_ITEMS do
      if Bench_Item_Stamp[i]>0 then begin
        BenchStop(i);
        writeln(BenchFile,' Total ',Bench_Item_Title[i],
                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec RUNNING');
      end else
        writeln(BenchFile,' Total ',Bench_Item_Title[i],
```

```
                                ' Time =
',Bench_Item_Total[i]/100:1:2,' sec');
    writeln(BenchFile,'Benchmark Ended   for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
  end;
```

```
{-------------------------------------------------------

    BEAMIO.INC  ---  BEAM Miscellaneous I/O Routines

-------------------------------------------------------}

procedure Abort;
var
  ch: Char;
begin
  TextMode(C80);
  ExitBeam;
end;

function FnAbort: Boolean;
var
  ch: Char;
begin
  AbortFlag := false;
  if keypressed then begin
    Center('Do you really want to abort (Y or N)?',25);
    repeat
      ch := upcase(ReadKey);
    until ch in ['Y','N'];
    if ch = 'Y' then begin
       AbortFlag := true;
       BORDERSOK := false;
    end else begin
       AbortFlag := false;
       BORDERSOK := true;
    end;
    gotoxy(1,25);
    write(Blank);
  end;
  FnAbort := AbortFlag;
end;
```

```
procedure JbError;
begin
  Center('JB Board I/O Timeout Error. Aborting Beam
Program',24);
  WaitKey;
  ClrScr;
end;

procedure SetScreen;
begin
  if OpMode = GraphicsOp then begin
    if not GrInstalled then begin
      InstallGraphics;
      GrInstalled := true;
    end else SetGraphMode(GraphMode);
    SetTextStyle(DefaultFont,HorizDir,1);
    SetColor(White);
    SetActivePage(0);
    SetVisualPage(0);
    DirectVideo := false;
  end else begin
    TextMode(C80);
    DirectVideo := true;
  end;
end;

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;
```

```
{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetTimeHarry;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;    { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;    { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;    { get hours }
          Hour := port[$71];
```

```
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $07;    { get day }
              Day := port[$71];
              port[$70] := $0A;
              if (port[$71] and $80)=0 then begin
                port[$70] := $08;    { get month }
                Mon := port[$71];
                port[$70] := $0A;
                if (port[$71] and $80)=0 then begin
                  port[$70] := $09;    { get year }
                  Year := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then good :=
true;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
  Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetTimeStamp;
var
```

```
  tday,tmon,tyear,tsec,tminu,thour: str10;
  AmPm: Char;
begin
  GetTimeHarry;
  tmon := LZero(Mon);
  tday := LZero(Day);
  thour := LZero(Hour);
  tminu := LZero(Minu);
  FNTimeStamp := tmon + tday + thour + tminu;

  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  CurrentDate := tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

function TimeStamp: String;
begin
  GetTimeStamp;
  TimeStamp := CurrentDate + ' ' + CurrentTime;
end;
```

```
function ReadADC(Channel: Integer): Integer;
var
  count: LongInt;
  ADC,i: Integer;
  RetryCount : Integer;
begin
  RetryCount := 10;
repeat
  count := Timeout;
  port[LaserAddr+12] := Channel;
  { Must wait 10 us for EOC to go LO (286 @ 4us/loop). }

  i := 1;
  while i < 10 do inc(i);
  { With Release 3.1 system configuration this is a 50
us delay.
    Calculated using an oscilloscope by Paul Marigold on
July 5, 1988.
    The exact delay should be 8 cycles plus 2 u. }

  repeat   { then approx 110 us for conversion with
signal D7 HI }
    dec(count);
  until (port[LaserAddr+16] > 127) or (count = 0);
  ADC := port[LaserAddr+12];
  if count = 0 then dec(RetryCount);
until ((RetryCount = 0) and (Count = 0)) or
(port[LaserAddr+16]>127);

  if (RetryCount = 0) and (Count = 0) then JbError
  else ReadADC := ADC;
end;

function ReadLaserBoard(Channel: Integer): real;
var
   ADC1,ADC10,ADC100 : integer;
```

```
   ADC : real;
begin
   if Channel = SensorChannel[2] then begin
     ADC1 := ReadADC(Channel);
     ADC10 := ReadADC(Channel+8);
     ADC100 := ReadADC(Channel+10);
     ADC100 := 180 - ADC100;
     if ADC100 < 0 then ADC100 := 0;
     ADC := 255;
     if (ADC1 >= 20) and (ADC1 < 254) then       ADC :=
ADC1
     else if (ADC10 > 18) and (ADC10 < 254) then ADC :=
ADC10 / 10
     else                                        ADC :=
ADC100 / 100;
{}{   writeln('*** ADC1=',ADC1,' ADC10=',ADC10,'
ADC100=',ADC100,' ***'); }
   end else begin
      ADC := ReadADC(Channel);
   end;
   if PeakReading < trunc(ADC+0.5) then PeakReading :=
trunc(ADC+0.5);
{if Channel = SensorChannel[3] then writeln(' **
PEAKREADING=',peakreading:4:1);}
   ReadLaserBoard := ADC;
end;

procedure ReadTemperature(var Resist,Temper: Real);
var
  temp : real;
  It, ADC: Integer;
begin
{
  SUBROUTINE: READ voltage of thermistor on JB Board
  use 1/T, LOG(R) function TO linearize the
interpolation
}
```

```
  ADC := ReadADC(TempChannel);
  Resist := 20000.0 * (255-ADC) / (255+ADC);    { from
20K resistor on JB Board }
  if ADC>214 then begin
    Temper := 71;
    exit;
  end;
  It := 0;
  while (It <= TempCalPts) and (Resistance[It] > Resist)
do inc(It);
  if Resistance[It] = Resist then
    Temper := Temperature[It]
  else begin
    temp := (1/Temperature[It-1] - 1/Temperature[It]) /
      (ln(Resistance[It-1]) - ln(Resistance[It]));
    Temper := 1/ (1/Temperature[It-1] - temp *
      (ln(Resistance[It-1]) - ln(Resist)));
  end;
  Temper := Temper - 273.15;
end;
```

```
{-------------------------------------------------------

    BEAMSCAN.INC -----     BEAM Laser Scanning Routines
-------------------------------------------------------
procedure UserWait;
begin
   writeln;
   writeln;
   writeln('    ... press any key to continue ');
   repeat delay(100) until KeyPressed;
   key:=Readkey;
end;

 {                                    ************
                                      * MoveBeam *
                                      ************

  Routine to move mirrors to X,Y directly
  Note: X,Y can be any kind of variable, we change to
integers and get ADC% (for speed: later fixed addrs
routines for each)
}
procedure AxisMove(pos: LongInt;Strobe: Byte);
begin
  if pos > 65534 then pos := 65534
    else if pos < 0 then pos := 0;
  portW[LaserAddr] := pos;
  port[LaserAddr+2] := port[LaserAddr+2] and not Strobe;
 { x is bit 0, y is bit 1 }
  port[LaserAddr+2] := port[LaserAddr+2] or Strobe;
end;

procedure MoveInXAxis(x: LongInt);
var
  stepX: LongInt;
begin
  if abs(x - oldX) > SlewLimitMaxStep then begin
```

```
    if x - oldX > 0 then begin
      stepX := oldX + SlewLimitMaxStep;
      while stepX < x do begin
        AxisMove(stepX,1);
        inc(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepX := oldX - SlewLimitMaxStep;
      while stepX > x do begin
        AxisMove(stepX,1);
        dec(stepX,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end;
  end;
  AxisMove(x,1);
  oldX := x;
end;

procedure MoveInYAxis(y: LongInt);
var
  stepY: LongInt;
begin
  if abs(y - oldY) > SlewLimitMaxStep then begin
    if y - oldY > 0 then begin
      stepY := oldY + SlewLimitMaxStep;
      while stepY < y do begin
        AxisMove(stepY,2);
        inc(stepY,SlewLimitMaxStep);
        delay(SlewLimitDelay);
      end;
    end else begin
      stepY := oldY - SlewLimitMaxStep;
      while stepY > y do begin
        AxisMove(stepY,2);
        dec(stepY,SlewLimitMaxStep);
```

```
        delay(SlewLimitDelay);
      end;
    end;
  end;
  AxisMove(y,2);
  oldY := y;
end;

procedure MoveBeam(x,y: LongInt);
begin
  if x < 0 then x := 1;
  if x > 65565 then x := 65564;
  if y < 0 then y := 1;
  if y > 65565 then y := 65564;
  if PROFMOVE = true then MoveInYAxis(65000);
  MoveInXAxis(x);
  MoveInYAxis(y);
end;

function VerifyRealBeamFound(var BeamProfileVariables:
BeamProfileVarType;
  Sensor: Integer): boolean;
  { This procedure verifies the ADC signal was not a
glitch by profiling the suspected pinhole and making an
11 x 11 step scan.  If 10% of the ADC readings from this
test are above the truncation mark ( 2* real background
+ 3 ) then we have found our pinhole. }
const
  VColMax = 11;
  VRowMax = 11;
var
  Pass,row,col,delay,waste,numAboveTruncLevel: Integer;
  Xcorner,Ycorner: LongInt;
  ADC : real;
begin
  numAboveTruncLevel := 0;
  with BeamProfileVariables[Sensor] do begin
```

```
    TotalWeight     := 0;   { background noise is
subtracted at the end }
    SaturationCount := 0;
    HiPoints        := 0;
    Xcorner := Xpos - (VColMax + 1) * ScanStep div 2;
    Ycorner := Ypos - (VRowMax + 1) * ScanStep div 2;
    MoveBeam(Xcorner,Ycorner);    { position beam for
start of stepping }
    waste := 0;
    while waste < InitialDelay do inc(waste);
    PeakReading := 0;
    for col:=1 to VColMax do begin  { prefer to return
around edges after line }
      Xpos := Xcorner + col * ScanStep;
      Ypos := Ycorner;
      MoveBeam(Xpos,Ypos);        { execute stepping for
reach column }
      waste := 0;
      while waste < ColToColDelay do inc(waste);
{ do col delay }
      for row:=1 to VRowMax do begin
        inc(Ypos,ScanStep);
        MoveInYAxis(Ypos);       { execute row stepping
prior to read }
        waste := 0;
        while waste < RowToRowDelay do inc(waste);
{ do row delay }
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
{ take reading }
        if ADC < BkgndThreshold then inc
(numAboveTruncLevel);
      end; {for row}
    end; {for col}
    MaxSensorReading := trunc(PeakReading + 0.5);
    if (numAboveTruncLevel >= (0.1 * VRowMax * VColMax))
    then begin
       VerifyRealBeamFound := true;
```

```
    end
    else begin
       VerifyRealBeamFound := false;
    end;
  end;
end;


{
                                  ******************
                                  * GetBeamProfile *
                                  ******************

}

procedure SetupBeamProfile(var BeamProfileVariables:
BeamProfileVarType;
  Sensor: Integer;X,Y: Real;_ScanStep,_BkgndThreshold:
Integer);
begin
  with BeamProfileVariables[Sensor] do begin
    if X               >= 0 then Xpos           :=
trunc(X);
    if Y               >= 0 then Ypos           :=
trunc(Y);
    if _ScanStep       >= 0 then ScanStep       :=
_ScanStep;
    if _BkgndThreshold >= 0 then BkgndThreshold :=
_BkgndThreshold;
    InitialDelay    := 2500;
    RowToRowDelay   := 200;
    ColToColDelay   := 2000;
  end;
end;

procedure GetBeamProfile(var BeamProfileVariables:
BeamProfileVarType;
```

```
  Sensor: Integer);
var
  Pass,row,col,waste: Integer;
  ADC : Real;
  Xcorner,Ycorner: LongInt;
begin
  with BeamProfileVariables[Sensor] do begin
    XYweight        := 0;
    Xweight         := 0;
    Yweight         := 0;
    TotalWeight     := 0;   { background noise is
subtracted at the end }
    SaturationCount := 0;
    HiPoints        := 0;
    BeamFound       := false;
    for col:=1 to ColMax do ColWeight[col] := 0;
    for row:=1 to RowMax do RowWeight[row] := 0;
    Xcorner := Xpos - (ColMax + 1) * ScanStep div 2;
    Ycorner := Ypos - (RowMax + 1) * ScanStep div 2;
    MoveBeam(Xcorner,Ycorner);    { position beam for
start of stepping }
    waste := 0;
    while waste < InitialDelay do inc(waste);
    PeakReading := 0;
    Xpos := Xcorner + ScanStep;
    Ypos := Ycorner;
    MoveInXAxis(Xpos);
    for col:=1 to ColMax do begin
      waste := 0;
      while waste < ColToColDelay do inc(waste);
{ do col delay }
      for row:=1 to RowMax do begin
        inc(Ypos,ScanStep);
        MoveInYAxis(Ypos);       { execute row stepping
prior to read }
        waste := 0;
```

```
        while waste < RowToRowDelay do inc(waste);
{ do row delay }
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
{ take reading }
        TotalWeight := TotalWeight + ADC;
        ColWeight[col] := ColWeight[col] + ADC;
        RowWeight[row] := RowWeight[row] + ADC;
        if ADC < BkgndThreshold then begin
             ADC := 0
          end else begin
            inc(HiPoints);
            BeamFound := true; { VerifyRealBeamFound is
not used here. }
          end;
        if ADC > 210 then inc(SaturationCount);
        BeamMap[row,col] := ADC;
        XYweight := XYweight + ADC;
        Xweight := Xweight + (Xpos * ADC);
        Yweight := Yweight + (Ypos * ADC);
      end; {for row}
      Xpos := Xpos + (ColMax * ScanStep);
      MoveInXAxis(Xpos);                          { move
to far right  }
      Ypos := Ypos - (RowMax*ScanStep);
      MoveInYAxis(Ypos);                          { move
back up       }
      Xpos := Xpos - (ColMax*ScanStep) + ScanStep;
      MoveInXAxis(Xpos);                          { pos.
for next move }
    end; {for col}
    MaxSensorReading := trunc(PeakReading + 0.5);
  end;
end;
{
                              **********************
                              * DisplayBeamProfile *
                              **********************
```

```
}
procedure DisplayBeamProfile(var BeamProfileVariables:
BeamProfileVarType;
  Sensor: Integer);
var
  row,col,gx,gy,a,cc,cd: Integer;
  SensorMultiplier     : Real;
  ADC                  : Integer;
begin
  if Display then begin
    gotoxy(70,1);
    write('Sensor #',Sensor);
    { This section of code computes a multiplier to
assure the display
      of beam profiles from sensor one and two have the
same size circles. }
    if (Sensor=1) or (Sensor=2) then begin
      if SensorCalib[1,3] > SensorCalib[2,3] then begin
         if Sensor = 2 then SensorMultiplier := 1
         else SensorMultiplier := SensorCalib[1,3] /
SensorCalib[2,3];
       end else begin
         if Sensor = 1 then SensorMultiplier := 1
         else SensorMultiplier := SensorCalib[2,3] /
SensorCalib[1,3];
       end;
    end else SensorMultiplier := 1; { Sensor 3 }
    if BeamFound then begin
      Center('Found Sensor #'+IntStr(Sensor),25);
      ClearCol(ColMax); { wuz div 2);}
      for col:=1 to ColMax do begin   { prefer to return
around edges after line }
        ClearCol((ColMax-col)); {wuz div 2); }
        for row:=1 to RowMax do begin
          gx := ((ColMax - col) * 3 + 1) * 8; { wuz 4; }
          gy := (RowMax - row + 1) * 14;      { wuz 7; }
```

```
          ADC := trunc(BeamProfileVariables[Sensor].
BeamMap[row,col]
                                * SensorMultiplier);
          a  := (ADC - trunc(Bk[Sensor]-3)) div
DiscreteBeam + 1;
          if a > 15 then a := 15;   { wuz 7--- maximum
size of circle }
          if OpMode = TextOp then begin
             { need mod 2 here to alleviate text
overwrites if ((col mod 2)=1) and ((row mod 2)=1) then
begin }
           {wuz this--  gotoxy(((ColMax - col) * 3 + 1)
div 2,(RowMax - row + 1) div 2);}
                gotoxy(((ColMax - col) * 3 + 1),(RowMax
- row + 1));
                if ADC < 1 then write('  +') else
write(ADC:3);
           {  end; this is all regarding hi-resolution
graphics }
          end else begin
            if a < 2 then PutPixel(gx,gy,3)
            else Circle(gx,gy,a);
          end;
          if keypressed then exit;
        end; {for row}
      end; {for col}
    end else Center('Sensor #'+IntStr(Sensor)+' not
found',25);
  end; {if Display}
end;

function BkReadFunction(Sensor: Integer;
XLocation,YLocation: LongInt): Real;
{  At last, a procedural way of reading background
noise. }
var
  waste,icount : LongInt;
```

```
  LoopStop        : Integer;
  hour,min,sec,sec100        : word;
  SummationADC, ADC, BkRead : real;
const
  InitialDelay    = 2500;
begin
  MoveBeam(XLocation,YLocation);
  waste := 0;
  while waste < InitialDelay do inc(waste);        { keep
the initial delay }
repeat
  icount := 0;
  SummationADC := 0;
  GetTime(hour,min,sec,sec100); { sec100 goes 0..99 }
  if (sec100 + 33) > 99 then LoopStop := sec100 + 33 -
99
                        else LoopStop := sec100 + 33;
  repeat
     inc(icount);
     ADC := ReadLaserBoard(SensorChannel[Sensor]);
     GetTime(hour,min,sec,sec100);
     SummationADC := SummationADC + ADC;
  until (sec100 >= LoopStop);
until (icount > 10);
  BeamProfileVariables[Sensor].AveBackground :=
SummationADC / icount;
  BkRead :=
trunc((BeamProfileVariables[Sensor].AveBackground
                                          * RowMax *
ColMax) + 0.5);
  SensorBkgnd[Sensor] := BkRead;       { subtract this
from the power array }
  Bk[Sensor] := (2.0 * BkRead) / (ColMax * RowMax) + 3;
{ truncation value }
  if Sensor = 3 then Bk[Sensor] := (1.25 * BkRead) /
(ColMax * RowMax) + 3;
  BkReadFunction := BkRead;
```

```
end;

{
                                    ************
                                    * GoodRead *
                                    ************
}

procedure GoodRead(iter: Integer);
var
  Try: Integer;
begin
  with BeamProfileVariables[Sensor] do begin
    Xaverage  := 0;
    Yaverage  := 0;
    TotWtAvg  := 0;
    GoodCount := 0;

    {   Re-read background noise level for increased
accuracy HERE.  Do not do it at every iteration because
it takes too long.     }

    if Sensor = 1 then
    SensorBkgnd[Sensor]  := BkReadFunction

(Sensor,trunc(Xsp[Sensor]),trunc(Ysp[Sensor]-2000));
{ true background }

    if Sensor = 2 then
    SensorBkgnd[Sensor]  := BkReadFunction
          (Sensor,trunc(Xsp[Sensor]),trunc(Ysp[Sensor]+
2000));  { true background }

    if Sensor = 3 then
    SensorBkgnd[Sensor]  := BkReadFunction
          (Sensor,Xrest,Yrest);  { true background }
```

```
    BkgndThreshold        := trunc(Bk[Sensor]+0.5);
{ high level mark }

    for Try := 1 to iter do begin
      SetupBeamProfile(BeamProfileVariables,Sensor,
Xsp[Sensor],
        Ysp[Sensor],4{ScanStep-wuz2},-1
{BkgndThreshold});
      GetBeamProfile(BeamProfileVariables,Sensor);
      if Display then
DisplayBeamProfile(BeamProfileVariables,Sensor);
      if XYweight > 0 then begin
        Xaverage := Xweight / XYweight + Xaverage;
        Yaverage := Yweight / XYweight + Yaverage;
        TotWtAvg := TotWtAvg + TotalWeight -
SensorBkgnd[Sensor];
        { TotWtAvg is the sum, TotalWeight is the
TotalArea under the profile curve, and SensorBkgnd
[Sensor] is the total background noise. }
        inc(GoodCount);
      end;
    end; {for Try}
    if GoodCount > 0 then begin
      Xaverage := Xaverage / GoodCount;
      Yaverage := Yaverage / GoodCount;
      TotWtAvg := TotWtAvg / GoodCount;
    end;
    Power[Sensor] := TotWtAvg * SensorCalib[Sensor,3];
  end;
end;

{-------------------------------------------------------

                                        ***********
                                        * IOSetup *
                                        ***********

}
```

```
procedure IOSetup;
begin
  Center('--- Hardware Reset ---',23);
  { control byte of Laser Board; A,B out, C 1/2 in 1/2
out and Thermistor Board same }
  port[LaserAddr+3] := $88;
  Xpos := XRest;
  Ypos := YRest;
  MoveBeam(Xpos,Ypos);
end;

{
                                    ************
                                    * FindBeam *
                                    ************
}

procedure SetupFindBeam(var FindBeamVars:
FindBeamVarType;
  Sensor: Integer;X,Y:
Real;_SpiralSideMax,_SpiralTries,_ScanStep,
  _BkgndThreshold,_SpiralDelay,_NumCenterPasses:
Integer);
begin
  with FindBeamVars[Sensor] do begin
    if X                >= 0 then Xval            :=
trunc(X);
    if Y                >= 0 then Yval            :=
trunc(Y);
    if _SpiralSideMax   >= 0 then SpiralSideMax   :=
_SpiralSideMax;
    if _SpiralTries     >= 0 then SpiralTries     :=
_SpiralTries;
    if _ScanStep        >= 0 then ScanStep        :=
_ScanStep;
```

```
    if _SpiralDelay     >= 0 then SpiralDelay     :=
_SpiralDelay;
    if _BkgndThreshold  >= 0 then BkgndThreshold  :=
_BkgndThreshold;
    if _NumCenterPasses >= 0 then NumCenterPasses :=
_NumCenterPasses;
  end;
end;

function SpiralSearchForHole(MssgKey:integer;var
FindBeamVars: FindBeamVarType;
  Sensor: Integer): Boolean;
  { MssgKey   2 = no verify on the searches }
var
  i,SpiralSide,waste: Integer;
  X,Y: LongInt;
  ADC : Real;
  junkkey : char;
begin
  with FindBeamVars[Sensor] do begin
    SetupBeamProfile(BeamProfileVariables,Sensor,Xsp
[Sensor],
    Ysp[Sensor],4{ScanStep},-1{BkgndThreshold});
    X := trunc(Xval);
    Y := trunc(Yval);
    MoveBeam(X,Y);     { starting point of box search }
    delay(SpiralDelay);
    BeamFound := false;
    SpiralSide := 1;
    while (SpiralSide <= SpiralSideMax) and not
BeamFound do begin
      i := -3;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInXAxis(X);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
```

```
        if ADC > BkgndThreshold then begin
           BeamProfileVariables[Sensor].Xpos := X;
           BeamProfileVariables[Sensor].Ypos := Y;
           if MssgKey <> 2 then begin
              if VerifyRealBeamFound(BeamProfile
Variables,Sensor)
              then BeamFound := true;
           end else BeamFound := true;
        end;
        if not BeamFound then begin
           if ( X + ScanStep < 65535 ) and ( X +
ScanStep > 0 )
           then inc(X,ScanStep);
        end;
        inc(i);
      end;
      i := -3;
      while (i <= SpiralSide) and not BeamFound do begin
        MoveInYAxis(Y);
        waste := 0;
        while (waste < SpiralDelay) do inc(waste);
        ADC := ReadLaserBoard(SensorChannel[Sensor]);
        if ADC > BkgndThreshold then begin
           BeamProfileVariables[Sensor].Xpos := X;
           BeamProfileVariables[Sensor].Ypos := Y;
           if VerifyRealBeamFound(BeamProfileVariables,
Sensor)
           then BeamFound := true;
        end;
        if not BeamFound then begin
           if ( Y + ScanStep < 65535 ) and ( Y +
ScanStep > 0 )
           then inc(Y,ScanStep);
        end;
        inc(i);
      end;
      if FnAbort then exit;
```

```
      { reverse directions after each 2 sides }
      if not BeamFound then ScanStep := -ScanStep;
      inc(SpiralSide);
    end; {while SpiralSide}
    if BeamFound then begin
      Xval := X;
      Yval := Y;
    end;
    SpiralSearchForHole := BeamFound;
  end;
end;

function FindBeam(var FindBeamVars:
FindBeamVarType;Sensor: Integer;
  FindMethod: FindMethodType; MssgKey: Integer):
Boolean;
  { MssgKey 0 =  no messages, 1 = messages, 2 = no
messages, no verify
    in SpiralSearch }
const
  HoleSeparation = 600;
var
  DisplayFlag: Boolean;
  i,pass,CantFind,SpiralSide: Integer;
  XspAv,YspAv,ADC: Real;
  X,Y,GoodX,GoodY: LongInt;
  BeamFound: Boolean;
begin
  FindBeam := true;
  DisplayFlag := Display;
  with FindBeamVars[Sensor] do begin
    CantFind  := 0;
    repeat
      X := trunc(Xval);
      Y := trunc(Yval);
      MoveBeam(X,Y);   { gets us to best starting
position }
```

```
      if MssgKey = 1 then begin
         FlushKey;
         Center('Searching for Sensor
#'+IntStr(Sensor)+': press any key to abort',25);
      end;
{
  now spiral to search for holes
}
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
        -1{SpiralTries},-1{ScanStep},-1
{BkgndThreshold},-1{SpiralDelay},
        -1{NumCenterPasses});
      BeamFound := SpiralSearchForHole(MssgKey,
FindBeamVariables,Sensor);
      if AbortFlag then exit;
      if not BeamFound then begin
        if MssgKey = 1 then
           Center('Can''t find Sensor #'+IntStr
(Sensor)+'. Is beam blocked?',24);
        SetupFindBeam(FindBeamVariables,Sensor,X,Y,
SpiralSideMax * 2,
          -1{SpiralTries},-1{ScanStep},BkgndThreshold -
1,-1{SpiralDelay},
          -1{NumCenterPasses});
        inc(CantFind);
      end;
    until (CantFind >= SpiralTries) or BeamFound;
    if CantFind >= SpiralTries then begin
      MoveBeam(Xval,Yval);
      if MssgKey = 1 then begin
         beep;
         Center('Could not find Sensor
#'+IntStr(Sensor),24);
         if not Calibrate then WaitKey;
      end;
      FindBeam := false;
```

```
      exit;
    end;
    X := Xval;
    Y := Yval;
{
  Verify that we have 2 mil hole...

  1/2 mil    1 mil ) left rear              2 mil     4
mil ) right front
                   ) of vat
  ) of vat
   4  mil    2 mil )                        1 mil    1/2
mil )

     Sensor #1                                  Sensor #2
```

We have the position of one of the four holes: 1/2, 1, 2, or 4 mils. We want to find the 2 mil hole closest to the vat. Note that the orientation of the holes is different for Sensors 1 and 2.

Hole Size Check Procedure:

1. We may have the position for the 1/2 or 1 mil hole. Move towards the vat in the X-axis and search for the 1 or 2 mil hole. If the beam is seen, adopt the new coordinates.

2. Now we may have the position for the 4 mil hole. Move towards the vat in the Y-axis and search for the 2 mil hole. If the beam is seen, use the new coordinates.

```
}
    if (Sensor < 3) and (FindMethod = Verify2MilHole)
then begin
      GoodX := X;
      GoodY := Y;
      if X < 32767 then inc(X,HoleSeparation)
```

```
        else dec(X,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,
SpiralSideMax div 2,
        -1{SpiralTries},4{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(0,FindBeamVariables,Sensor)
then begin
          GoodX := Xval;
          GoodY := Yval;
        end;
      X := GoodX;
      Y := GoodY;
      if AbortFlag then exit;
      if Y < 32767 then inc(Y,HoleSeparation)
        else dec(Y,HoleSeparation);
      SetupFindBeam(FindBeamVariables,Sensor,X,Y,-1
{SpiralSideMax},
        -1{SpiralTries},4{ScanStep},-1{BkgndThreshold},
-1{SpiralDelay},
        -1{NumCenterPasses});
      if SpiralSearchForHole(0,FindBeamVariables,Sensor)
then begin
        GoodX := Xval;
        GoodY := Yval;
      end;
      X := GoodX;
      Y := GoodY;
      if AbortFlag then exit;
    end;
       if DisplayFlag then begin
      SetScreen;
      gotoxy(70,1);
      write('Sensor #',Sensor);
      if BeamFound then Center('Found Sensor
#'+IntStr(Sensor),25)
      else if MssgKey = 1 then
```

```
        Center('Sensor #'+IntStr(Sensor)+' not
found',25);
    end;
    XspAv := 0;
    YspAv := 0;
    if BeamFound then begin
      for pass:=1 to NumCenterPasses do with
BeamProfileVariables[Sensor] do begin
        CantFind := 0;
        { discard the first }

  { Bk[Sensor] is assigned to BeamProfilVariables
[Sensor].BkgndThreshold HERE. }
        SetupBeamProfile(BeamProfileVariables,
Sensor,X,Y,
          4{ScanStep-wuz2},trunc(Bk[Sensor] + 0.5));
        repeat
          GetBeamProfile(BeamProfileVariables,Sensor);
          if XYweight = 0 then inc(CantFind);
        until (XYweight > 0) or (CantFind > 3);

        { scanstep is reset to four HERE, scanstep of 2
is required for GetBeamProfile
        BeamProfileVariables[Sensor].scanstep := 4;
wuz2 }

        if CantFind > 3 then begin
          if MssgKey = 1 then begin
             beep;
             Center('Can''t profile Sensor
#'+IntStr(Sensor)+'. Try (6) Move',24);
             if not Calibrate then WaitKey;
          end;
          FindBeam := false;
          exit;
        end;
```

```
        if (SaturationCount > 0) and (MssgKey = 1) then
begin
          gotoxy(1,24);
          Center('WARNING: '+IntStr(SaturationCount)
            + ' sensor readings near saturation',24);
          delay(1000);
        end;
{
  Later select sensor hole size automatically
}
        X := trunc(Xweight / XYweight);   { better
coordinates }
        Y := trunc(Yweight / XYweight);
        Power[Sensor] := (TotalWeight -
SensorBkgnd[Sensor])
          * (SensorCalib[Sensor,3]);
        { POWER = measure of beam power profile - actual
background noise
                  ( times ) the characteristice factor
for sensors }
        XspAv := XspAv + Xweight / XYweight;
        YspAv := YspAv + Yweight / XYweight;
      end; {for pass}
      if NumCenterPasses > 1 then begin
        XspAv := XspAv / NumCenterPasses;
        YspAv := YspAv / NumCenterPasses;
      end;
    end; {if BeamFound}
    Xval := X;        { update integer coordinates }
    Yval := Y;
    Xavg := XspAv;    { update average coordinates }
    Yavg := YspAv;
    if DisplayFlag then begin
      DisplayBeamProfile(BeamProfileVariables,Sensor);
      PrintInfo(Sensor);
    end;
  end;
```

```
end;

function CoarseFindBeam(Sensor: Integer): Boolean;
var
  OldDisplay: Boolean;
begin
  OldDisplay := Display;
  Display := false;
  { Bk[Sensor] := 2 * (background noise level) + 3 }
  SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
    50{50 sides of 16(ScanStep) each= 800 true length of
SpiralSideMax},
    3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] + 0.5){BkgndThreshold},1
{SpiralDelay},1{NumCenterPasses});
  if not FindBeam(FindBeamVariables,Sensor,NormalFind,0
{no mssgs})then begin
     SetupFindBeam(FindBeamVariables,Sensor,Xsp[Sensor],
Ysp[Sensor],
       100{800*1.5=1200= true SpiralSideMax
(1200/12)},3{SpiralTries},
       12{16/1.5=ScanStep},trunc(Bk[Sensor]+
0.5){BkgndThreshold},
       1{SpiralDelay},1{NumCenterPasses});
     if not FindBeam(FindBeamVariables,Sensor,
NormalFind,0{no mssgs}) then begin
        { Resetting Bk[Sensor] much lower always yields
beam found.}
        SetupFindBeam(FindBeamVariables,Sensor,
Xsp[Sensor],Ysp[Sensor],
          225{1200*1.5=1800=true SpiralSideMax
(1800/8)},3{SpiralTries},
          8{12/1.5=ScanStep},trunc(Bk[Sensor] +
0.5){BkgndThreshold},
          1{SpiralDelay},1{NumCenterPasses});
```

```
        if not FindBeam(FindBeamVariables,Sensor,
NormalFind,1{messages}) then begin
            CoarseFindBeam := BeamFound;
            exit;
        end;
      end;
  end;
  Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
  Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  Display := OldDisplay;
  CoarseFindBeam := BeamFound;  { case of BeamFound =
true }
end;

function FineFindBeam(Sensor: Integer): Boolean;
begin
  SetupFindBeam(FindBeamVariables,Sensor,
Xsp[Sensor],Ysp[Sensor],
    50{SpiralSideMax},3{SpiralTries},16{ScanStep},
    trunc(Bk[Sensor] +
0.5){BkgndThreshold},100{SpiralDelay},
    2{NumCenterPasses});
  { Bk[Sensor is assigned to FindBeamVars[Sensor].
BkgndThreshold HERE. }
  if FindBeam(FindBeamVariables,Sensor,Verify
2MilHole,1{messages}) then begin
    Xsp[Sensor] := FindBeamVariables[Sensor].Xval;
    Ysp[Sensor] := FindBeamVariables[Sensor].Yval;
  end;
  FineFindBeam := BeamFound;
end;

function FindLaserBeam(Sensor: Integer): Boolean;
begin
  if CoarseFindBeam(Sensor) and not AbortFlag then
    BeamFound := FineFindBeam(Sensor);
  FindLaserBeam := BeamFound;
```

```
end;

function FindHoles2(Sensor: Integer): Boolean;
{ This procedure re-establishes the sensor coordinates.
}
begin
  FindHoles2 := FindLaserBeam(Sensor);
end;

function FindHoles(Sensor: Integer): Boolean;
{
  Updates hole position
  Needs to confirm that we have 2 mil hole
}
var
  BkRead: Real;
  row,col: Integer;
  ADC : real;
begin
  FlushKey;
  Center('Searching for Sensor #'+IntStr(Sensor)+':
press any key to abort',25);

  with BeamProfileVariables[Sensor] do begin
  {   Re-read background noise level for increased
accuracy HERE.  Do not do it at every iteration because
it takes too long.     }

    SensorBkgnd[Sensor]  :=
BkReadFunction(Sensor,Xrest,Yrest);  { true background }
    BkRead               := SensorBkgnd[Sensor];
    BkgndThreshold       := trunc(Bk[Sensor]+0.5);
{ high level mark }
  end;

  FindHoles := FindHoles2(Sensor);
```

```
  { FindHoles2 goes through a coarse and fine beam
location alogorithm.    }
end;
```

```
{-------------------------------------------------------

     BEAMUTIL.INC ------      BEAM Utility Routines

-------------------------------------------------------}

procedure beep;
begin
  Sound(1000);
  delay(300);
  NoSound;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

procedure centerln(text: String);
begin
  gotoxy((79-length(text)) div 2,whereY);
```

```
  writeln(text);
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
  Str(num,temp);
  LongStr := temp;
end;

function RealStr(num: Real;AfterDecimal: Integer):
String;
var
  i: Integer;
  temp: String;
begin
  Str(num:10:AfterDecimal,temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
```

```
end;

function spc(len: Integer): string;
begin
  spc := copy(blank,1,len);
end;

procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: String): String;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
```

```
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;

{
  Find any char of Chars in Main starting at index Start

  Returns length(Main)+1 if no character of Chars in
Main
}
function FindStr(Start: Integer;Main: string;Chars:
str16): Integer;
var
  i,j,min: Integer;
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
  for i:=1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
```

```
end;

function WordVal(line: String;var code: Integer): Word;
var
  code2: Integer;
  num: Word;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  LongVal := num;
end;

function RealVal(line: String;var code: Integer): Real;
var
  ofs,code2: Integer;
  num: Real;
begin
  ofs := 0;
  if line[1]='.' then begin
    line := '0' + line;
    ofs := -1;
  end;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  if code<>0 then inc(code,ofs);
  RealVal := num;
```

```
end;

{
  Pick out a number from a string starting at Index and
going to the next space or comma or end-of-string
}
procedure UnitValShortInt(var Storage: Integer;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure UnitValInt(var Storage: LongInt;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := LongVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

procedure UnitVal(var Storage: Real;Text: String;var
Index: Integer);
var
  i: Integer;
begin
  Storage := RealVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else inc(Index,i);
end;

{
```

```
  Pick out text from a string starting at Index and
going to the next comma or end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
Integer);
var
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

{
  Show error and store it on disk
}
procedure QuietSystemError(TextLine: string);
begin
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFile,'BEAM: ',TextLine);
  close(ErrorFile);
end;

procedure SystemError(TextLine: string);
begin
  writeln(TextLine);
  QuietSystemError(TextLine);
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;
```

```
{ Harry's ultimate Draw-A-Box on the text screen routine
}
procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
  const
                    {  Type          1     2     3     4
}
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then begin
    if y1<>y2 then FullHorzBar :=
Many(HorzBar[BoxType],x2-x1+1)
      else FullHorzBar :=
Many(HorzBar[BoxType],x2-x1-1);
    end else FullHorzBar := '';
  if y1<>y2 then begin
    if x1<>x2 then write(UpperLeftCorner[BoxType]);
    write(FullHorzBar);
```

```
    if x1<>x2 then write(UpperRightCorner[BoxType]);
  end;
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
  if y1 = y2 then write(FullHorzBar)
  else begin
    if x1<>x2 then write(LowerLeftCorner[BoxType]);
    write(FullHorzBar);
    if x1<>x2 then write(LowerRightCorner[BoxType]);
  end;
end;

{$F+}
procedure ExitBeam;
{$F-}
begin
  TextMode(C80);
  Exec('C:\COMMAND.COM','/C SERVICE LASER OFF +60/i/t');
(*
  if ExitCode > 0 then begin
    write('exit code ',ExitCode,' at addr ',Seg
(ErrorAddr));
    key := ReadKey;
  end;
*)
  ExitProc := OldExitProc;
end;

procedure ClearLine(row: Integer);
begin
  gotoxy(1,row);
  write(blank);
```

```
end;

procedure Center(Text: String;Line: Integer);
begin
  gotoxy(1,Line);
  write(Blank);
  gotoxy((79-length(Text)) div 2,Line);
  write(Text);
end;

procedure PrintXY(x,y: Real);
var
  s: String;
begin
  if x < 0     then x := 0;
  if x > 65535 then x := 65535;
  if y < 0     then y := 0;
  if y > 65535 then y := 65535;
  s := '(' + WordStr(trunc(x)) + ',' + WordStr(trunc(y))
+ ')';
  if length(s)<13 then s := spc((13-length(s)) div 2) +
s;
  write(s,spc(13-length(s)));
end;

procedure PrintPowerPos(Sensor: Integer);
begin
  ClearLine(23);
  gotoxy(10,23);
  write('Sensor #',Sensor,'Power',Power[Sensor]:5:1,'
mW, Center');
  PrintXY(Xsp[Sensor],Ysp[Sensor]);
end;

procedure PrintInfo(Sensor: Integer);
  var
    numreadings,jj : integer;
```

```
    AverageBeamPower : real;
begin
  numreadings := 0;
  { numreadings is also the location of the first zero
entry }
  AverageBeamPower := 0;
  repeat
     numreadings := numreadings + 1;
  until (numreadings = 9) or (AverageBeamPowerArray
[numreadings] = 0);
  if numreadings = 9 then begin
  { Now AveBeamPowerArray[9 or 10] could be zero. }
     if (AverageBeamPowerArray[9] <> 0) then begin
        numreadings := 10;
        if (AverageBeamPowerArray[10] <> 0) then for jj
:= 1 to 9 do
                    AverageBeamPowerArray[jj] :=
AverageBeamPowerArray[jj+1];
     end;
  end;
  { add on most recent value }
  AverageBeamPowerArray[numreadings] := Power[Sensor];
  for jj := 1 to numreadings do
           AverageBeamPower := AverageBeamPower +
                               AverageBeamPowerArray
[jj];
  AverageBeamPower := (AverageBeamPower+Power[Sensor]) /
(numreadings+1);
  { Weight the last reading double and divide by one
extra.  This is chosen over a longer average without an
array on purpose, ie. subtract out the average when
loping off the, say, hundredth reading.  }

  gotoxy(71,3);
  write('Power');
  gotoxy(69,4);
  write('(average)');
```

```
    gotoxy(69,5);
    write(AverageBeamPower:5:1,' mW');
    gotoxy(71,7);
    write('Power');
    gotoxy(68,8);
    write('(immediate)');
    gotoxy(69,9);
    write(Power[Sensor]:5:1,' mW');
    gotoxy(71,11);
    write('Center');
    gotoxy(68,12);
    PrintXY(Xsp[Sensor],Ysp[Sensor]);
    gotoxy(70,15);
    write('Summation');
    gotoxy(70,16);
    write(BeamProfileVariables[Sensor].TotalWeight:6:0);
    gotoxy(71,18);
    write('Maximum');
    gotoxy(73,19);
    write(BeamProfileVariables[Sensor].MaxSensorReading);
    BeamProfileVariables[Sensor].MaxSensorReading := 0;
end;

procedure WaitKey;
begin
  Center('<Hit any key to continue>',25);
  repeat
  until keypressed;
  gotoxy(25,1);
  write(blank);
end;

procedure Pause(usec: LongInt);
var
  i,msec: LongInt;
begin
  i := 1;
```

```
  key := ' ';
  msec := trunc(round(usec / 1000));
  while (i < msec) and not keypressed do begin
    delay(1);
    inc(i);
  end;
  if keypressed then key := ReadKey;
end;

procedure FlushKey;
begin
  while keypressed do key := ReadKey;
end;

procedure ClearCol(col: Integer);
var
  col2: Integer;
begin
  if (col > 0) and (OpMode = GraphicsOp) then begin
    col2 := (col-1)*24-4;
    if col2<0 then col2 := 0;
    SetFillStyle(EmptyFill,0);
    Bar(col2,0,col2+23,330);
  end;
end;

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;
```

```
procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;
```

```
procedure BenchInit;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    for i:=1 to BENCH_ITEMS do begin
      Bench_Item_Total[i]:=0;
      Bench_Item_Stamp[i]:=0;
    end;
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    writeln(BenchFile,'Benchmark Started for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
    Bench_Start_DOW := DOW;
  end;


function GetS100Time : LongInt;
  var
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    if DOW<Bench_Start_DOW then Inc(DOW,7);
    GetS100Time := S100 + LongInt(Sec)*100 +
LongInt(Min)*6000 + Hr*360000 +
                   (DOW-Bench_Start_DOW)*8640000;
  end;
```

```
procedure BenchStart(Item : Word);
  begin
    if Bench_Item_Stamp[Item]=0 then
Bench_Item_Stamp[Item]:=GetS100Time;
  end;

procedure BenchStop(Item : Word);
  begin
    if Bench_Item_Stamp[Item]>0 then begin
      Inc(Bench_Item_Total[Item],GetS100Time -
Bench_Item_Stamp[Item]);
      Bench_Item_Stamp[Item]:=0;
    end;
  end;

procedure BenchEnd;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    for i:=1 to BENCH_ITEMS do
      if Bench_Item_Stamp[i]>0 then begin
        BenchStop(i);
        writeln(BenchFile,' Total ',Bench_Item_Title[i],
                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec RUNNING');
      end else
        writeln(BenchFile,' Total ',Bench_Item_Title[i],
                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec');
```

```
    writeln(BenchFile,'Benchmark Ended   for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
  end;
```

```
{-------------------------------------------------------

   BEAMGR.INC -----          BEAM Graphics Routines

-------------------------------------------------------}

procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn, 0, ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x+TextWidth(Text),y+TextHeight(Text));
  SetColor(color);
  OutTextXY(x,y,Text);
end;

function GetInput(maxlen: Integer): String;
var
  ch: Char;
  temp: String;
  input: Boolean;
  baseX,baseY,index: Integer;
begin
  temp := '';
  index := 0;
```

```
    baseX := getX;
    baseY := getY;
    key := chr(0);
    while key<>Cr do begin
      MoveTo(baseX+index*8,baseY);
      SetColor(3);
      OutText(chr(219));
      key := ReadKey;
      if key = CtrlC then Abort;
      MoveTo(baseX+index*8,baseY);
      SetColor(0);
      OutText(chr(219));
      MoveTo(baseX+index*8,baseY);
      SetColor(3);
      if key in [Bs,Cr,' '..'~'] then begin
        if key=Bs then begin
          if index>0 then dec(index);
        end else if key in[' '..'~'] then begin
          OutText(key);
          if index<maxlen then inc(index);
          temp[index] := key;
        end;
      end;
    end;
    temp[0] := chr(index);
    GetInput := temp;
    SetColor(3);
  end;

  procedure GrAbort(Msg: string);
  begin
    Writeln(Msg,': ',GraphErrorMsg(GraphResult));
    Halt(1);
  end;

  procedure InstallGraphics;
  begin
```

```
  { Register all the drivers }
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    GrAbort('EGA/VGA');

  GraphDriver := Detect;                { autodetect the
hardware }
  InitGraph(GraphDriver,GraphMode,'');  { activate
graphics }
  if GraphResult<>grOk then             { any errors? }
  begin
    Writeln('Graphics init error: ',
GraphErrorMsg(GraphDriver));
    Halt(1);
  end;
  SetVisualPage(0);
  SetActivePage(0);
  ClearDevice;
end;

{
  Print action information
}
procedure Action(Text: String);
var
  i: Integer;
begin
  if OpMode=TextOp then write(Text,' ')
  else begin
    MoveTo(40*8,7*14);
    SetColor(2);
    OverText(Text,20);
    SetColor(3);
  end;
end;

procedure Message(Text: String);
begin
```

```
  MoveTo(3*8,24*14-2);
  OverText(Text,length(Text)+5);
end;
```

```
{=============================================================
   3D Systems Stereolithography System Software
=============================================================

   COMBINE.INC
   SLA-1 This is the master include library to COMBINE
shared procedures

   The following is a list and description of the
subroutines:

   CursorOff - remove the cursor from the SLA display
screen
   CursorOn  - show the cursor on the SLA display screen
   Beep      - produce a auditable beep

   Upper     - convert a string to all upper case
   Many      - propigate a charater a specified number
of times in a string

   Drawbox   - draw a box of variable size which can be
used to border text

   Abort     - abort the current operation and return to
the previous task
   HandleKeyboardRequest - Read the key unless it is a
cntl-c

   InitZTable - initialize the SLA platform
   ZTable     - access the port for the SLA platform
   ZMove      - send a command to the Z-axis stepper
controller
   FormatZCommand - move the platform a specified
distance
   WaitForIdleZStage - keep the platform moving until
the space bar is pressed
```

```
-------------------------------------------------------}

procedure CursorOff;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH or $20;
    Intr($10,regs);
  end;
end;

procedure CursorOn;
begin
  with regs do begin
    AH := 3;
    BH := 0;
    Intr($10,regs);
    AH := 1;
    CH := CH and $DF;
    Intr($10,regs);
  end;
end;

procedure beep;
begin
  sound(1000);
  delay(500);
  NoSound;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
```

```
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] := upcase
(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);
  IntStr := temp;
end;

function sgn(num: real): real;
begin
   if num<0 then sgn := -1
   else if num>0 then sgn := 1
   else sgn := 0;
end;

function RealStr(num: Real;AfterDecimal: Integer):
str80;
var
   i : integer;
   temp : str80;
```

```
begin
   Str(num:10:AfterDecimal,temp);
   i := 1;
   while (i<length(temp)) and (temp[i]=' ') do inc(i);
   RealStr := copy(temp,i,99);
end;

procedure Abort;
var
  key: Char;
begin
  while keypressed do key := ReadKey;
  CursorOn;
  halt(127);
end;

procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    key := ReadKey;
    if key = Ctrl C then Abort;
    if (key = #0) and keypressed then
      key := ReadKey;
  end;
end;

procedure DrawBox(x1,y1,x2,y2: integer;BoxType:
integer);
type
  BoxMode = array[1..4] of Char;
{
    Box Types:
      1:  Single Line all around
      2:  Double Line all around
      3:  Single Line horizontal, Double Line vertical
      4:  Double Line horizontal, Single Line vertical
}
```

```
  const
                        {  Type          1     2     3     4   }
    UpperLeftCorner : BoxMode = (#218,#201,#214,#213);
    UpperRightCorner: BoxMode = (#191,#187,#183,#184);
    LowerLeftCorner : BoxMode = (#192,#200,#211,#212);
    LowerRightCorner: BoxMode = (#217,#188,#189,#190);
    HorzBar         : BoxMode = (#196,#205,#196,#205);
    VertBar         : BoxMode = (#179,#186,#186,#179);
var
  i: integer;
  FullHorzBar: String;
begin
  gotoxy(x1,y1);
  if x1<x2 then FullHorzBar := Many(HorzBar[BoxType],
x2-x1-1)
    else FullHorzBar := '';
  if x1<>x2 then write(UpperLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(UpperRightCorner[BoxType]);
  for i:=y1+1 to y2-1 do begin
    gotoxy(x1,i);
    write(VertBar[BoxType]);
    gotoxy(x2,i);
    write(VertBar[BoxType]);
  end;
  gotoxy(x1,y2);
  if x1<>x2 then write(LowerLeftCorner[BoxType]);
  write(FullHorzBar);
  if x1<>x2 then write(LowerRightCorner[BoxType]);
end;

{-------------------------------------------------------

                        Z-Stage Elevator Routines

-------------------------------------------------------}
```

```
procedure InitZTable;
begin
  port[ZAddress+1] := $64;
  port[ZAddress+1] := $60;
  delay(100);
  port[ZAddress+1] := $40;
  port[ZAddress+1] := $60;
  delay(100);
end;

 { procedure ZTable(Ch: Char);
var
  b: Integer;
begin
  repeat
    b := port[ZAddress+1];
  until (b and $10)>0;
  port[ZAddress] := ord(Ch);
  port[ZAddress+1] := $70;
  repeat
    B:=port[ZAddress+1];
  until (b and $10)=0;
  port[ZAddress+1] := $60;
end;

procedure ZMove(Cmd: string);
begin
  if ShowStuff then begin
    if whereX>1 then writeln;
    writeln('   ZTable>',Cmd);
  end;
  if not ZTableOff then begin
    Cmd := Cmd + Cr;
    for i:=1 to length(Cmd) do ZTable(Cmd[i]);
  end;
end;
```

```
procedure FormatZCommand(Distance: Real);
begin
  Distance := sgn(Distance) * int(abs(Distance) * ZPitch
* 1000 / 25.4);
  if NegZStage then Distance := -Distance;
  ZMove('D'+RealStr(Distance,0)+' G');
end;

procedure WaitForIdleZStage;
var
  b: Word;
begin
  repeat
    b := port[ZAddress+1];
  until keypressed or (b and 2>0);
end;   ******** }
^Z
```

```
{
  STEREO Version 3.14 Binary Command Equates
}

const
  STEREO_BX = 1;
  STEREO_BY = 2;
  STEREO_JX = 3;
  STEREO_JY = 4;
  STEREO_NX = 5;
  STEREO_NY = 6;
  STEREO_SS = 7;
  STEREO_SP = 8;
  STEREO_SD = 9;
  STEREO_JP = 10;
  STEREO_JS = 11;
  STEREO_JD = 12;
  STEREO_LO = 13;
  STEREO_LF = 14;
  STEREO_NS = 15;
  STEREO_AB = 16;
  STEREO_EX = 17;
  STEREO_EC = 18;
  STEREO_CL = 19;
  STEREO_MD = 20;
  STEREO_SR = 21;
  STEREO_HR = 22;
  STEREO_RC = 23;
  STEREO_RD = 24;
  STEREO_RS = 25;
  STEREO_VC = 26;
  STEREO_VP = 27;
  STEREO_VR = 28;
  STEREO_VX = 29;
  STEREO_VY = 30;
  STEREO_WI = 31;
  STEREO_DC = 32;
```

```
STEREO_R1 = 33;
STEREO_R2 = 34;
STEREO_D1 = 35;
STEREO_D2 = 36;
STEREO_BG = 37;
STEREO_JM = 38;
STEREO_MV = 39;
STEREO_RV = 40;
```

```
{
=============================================================
```

StereoLithography Laser Controller Software Interrupt Interface Routines

for 3D Systems, Inc.

Version 2.80 by Tarnz Technologies

```
=============================================================
```

The purpose of these routines is to interface general-purpose Turbo Pascal programs with the memory resident laser controller software. The machine language-based laser controller software the routines below interface to can be used with other languages like Basic, C, and 8086 Assembly Language. Turbo Pascal was chosen because of its development environment and low source-to-executable-object code turnaround time. As development progresses and there is no longer a great need for a fast development environment, these routines should be adapted to C to maintain project software consistency. Turbo C and Microsoft C are the leading choices for C languages.

Below is a list of the routines in this include file. Most of the functions relate directly to specific software interrupt calls. The Code is the related interrupt's AX (AH,AL) value. Some of the interrupt functions require arguments to be passed in other registers, and some will return results. See the actual routines for more details on the interface protocol.

Pascal Function	Code	Description
__LaserOk		see if laser software is installed
__LaserPresent		tell if laser software not installed
__OpenLaser	00 --	open laser DAC channel and initialize
__CloseLaser	01 --	close laser DAC channel and restore system
	02 xx	add command to background command queue
__AddVector	02 00	add vector command
__AddOpenShutter	02 01	add open shutter command
__AddCloseShutter	02 02	add close shutter command
__AddChangeTimebase	02 03	add change time base command
__AddPause	02 04	add pause command
__ResetCmdQueue	03 --	reset background command queue
__GetCmdQueueSize	04 --	get background command queue size
	05 xx	get option
__GetIntNum	05 00	get software interrupt vector number
__GetBkgndTime	05 01	get background task interval time
__GetControlWord	05 02	get laser control word
__AutoShutterModeOn	05 02	get status of auto shutter mode
__PosAckModeOn	05 02	get status of position acknowledge mode
__BlockModeOn	05 02	get status of block mode

```
    __GetStatus           05 03  get laser status
    __GetOpenStatus       05 03  get laser channel open
status
    __GetBkgndIdleStatus  05 03  get background idle
status
    __GetTurnOffDelay     05 04  get laser turn-off delay
    __GetTurnOnDelay      05 05  get laser turn-on delay
    __GetHomePosition     05 06  get laser home position

                          06 xx  set option
    __SetIntNum           06 00  set software interrupt
vector number
    __SetBkgndTime        06 01  set background task
interval time
    __SetControlWord      06 02  set laser control word
(general)
    __SetAutoShutMode     06 02  set laser control word,
bit 0
    __SetPosAckMode       06 02  set laser control word,
bit 1
    __SetBlockMode        06 02  set laser control word,
bit 2
                          06 03  -
    __SetTurnOffDelay     06 04  set laser turn-off delay
    __SetTurnOnDelay      06 05  set laser turn-on delay
    __SetHomePosition     06 06  set laser home position

                          07 xx  background task
execution control
    __Execute             07 00  execute command queue to
current point and stop
    __ExecuteClear        07 01  execute command queue to
current point, stop, and clear
    __Clear               07 02  clear command queue
(reset queue's start pointer)
```

```
                          08 xx  background task redraw
control
    __SetStepPeriods      08 00  set multi-pass step
periods
    __SetRedrawCount      08 01  set redraw pass count
    __SetRedrawDelay      08 02  set redraw delay
    __Mark                08 03  mark redraw start
position
    __Redraw              08 04  perform multi-pass
redraw


=========================================================}

const
  _LASERVECTOR       =   $64;          { Interrupt
vector alternative  }
                                       { values: $60
through $67.      }
  _PACKET_SIZE       =    13;          { length of
vector packet       }

  _OK                =     0;          { Errors returned
by Level Zero }
  _INTERNAL_ERROR    =   255;
  _MINIMUM_TICKS     =    59;          { don't go below
50 us          }
  _LASER_QUEUE_MAX   = 60000.0;        { Laser Queue
size              }

  __ALLBITS = $FFFF;
  __BITVALUE: array[0..15] of Word =
    ($0001, $0002, $0004, $0008, $0010, $0020, $0040,
$0080, $0100, $0200, $0400, $0800, $1000, $2000, $4000,
$8000);

type
```

```
  _Str16  = string[16];
  _Str255 = string[255];
  _VectorType = record
      fractX: Byte;     { start coordinates   }
      startX: Word;
      fractY: Byte;     { fractional portion - 1 byte,
should be set to 0 }
      startY: Word;     { integer portion    - 2 bytes
                }
      deltaX: Integer;  { +/- change values in 1/256
units                }
      deltaY: Integer;
        incr: Word;     { # of increments within vector
count             }
      end;
  _StepPeriodsType = record
      num    : Byte;
      periods: array[0..9] of Word;
    end;

var
  __IntReg: Registers;
  LaserVersion: _Str16;

{-------------------------------------------------------

    __LaserOk function
      check to see if laser controller code has been
installed.

    returns:
    false if code not installed
    true  if code is installed
    LaserVersion contains laser controller version
number

-------------------------------------------------------}
```

```
function __LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with __IntReg do begin
      AX := ($35 shl 8) or _LASERVECTOR;
      Intr($21,__IntReg);
      CopyPtr := Ptr(Es,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      __LaserOk := true;
      with __IntReg do
        CopyPtr := Ptr(Es,BX-46);
      move(CopyPtr^[1],LaserVersion[1],18);
      LaserVersion[0] := #4;
    end else begin
      __LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

{-------------------------------------------------------

    __LaserPresent function
      same as LaserOk but also prints a message if laser
system is not installed.

    returns:
    false if code not installed
    true  if code is installed
```

```
-------------------------------------------------------}

function __LaserPresent: Boolean;
  begin
    if __LaserOk then __LaserPresent := true
    else begin
      __LaserPresent := false;
      writeln('LASER system not installed.');
    end;
  end;

{-------------------------------------------------------

    __OpenLaser(Ticks) function
      open up laser DAC channel with the background
server interrupting every Ticks timer 0 ticks.  Input
for timer 0 is a 1.19 MHz clock signal, so if Ticks =
119, the background task would activate once per 100 us;
with Ticks = 1190, once per ms;  etc.

    returns error code

-------------------------------------------------------}

function __OpenLaser(ticks: Word): Integer;
  begin
{ Min Time CHeck bypassed -- BE CAREFUL!!!!! }
{    if ticks<119 then ticks := 119; }
    with __IntReg do begin
      AX := $0000;
      BX := ticks ;    (* trunc(Period*1.19 {MHz}); *)
      Intr(_LASERVECTOR,__IntReg);
      __OpenLaser := AL;
    end;
  end;

{-------------------------------------------------------
```

```
    __CloseLaser function
      close off laser DAC channel after the background
command queue empties.  If laser needs to be stopped
immediately, __ResetVectorPool should be called before
this routine.

    returns error code

-------------------------------------------------------}

function __CloseLaser: Integer;
  begin
    with __IntReg do begin
      AX := $0100;
      Intr(_LASERVECTOR,__IntReg);
      __CloseLaser := AL;
    end;
  end;

{-------------------------------------------------------

    __AddVector(startX,startY,deltaX,deltaY,incr)
function
      add the vector to the background command queue.
Vector start coordinates are (startX,startY) and their
change value is (deltaX,deltaY).  The number of
incremental steps in the vector is incr.  deltaX and
deltaY are in a special format (high byte is +/- integer
portion; low byte is 1/256 units) conducive to high-
speed DAC updating.  See the 3DPC protocol specification
for more information.

    returns error code

-------------------------------------------------------}
```

```
function __AddVector(startX,startY: Word;deltaX,deltaY:
Integer;
                        incr: Word): Integer;
  var
    VectorPacket: _VectorType;
  begin
    if incr>0 then begin
      VectorPacket.fractX := 0;       { set up vector
packet }
      VectorPacket.startX := startX;
      VectorPacket.fractY := 0;
      VectorPacket.startY := startY;
      VectorPacket.deltaX := deltaX;
      VectorPacket.deltaY := deltaY;
      VectorPacket.incr   := incr;
      with __IntReg do begin
        AX := $0200;
        ES := Seg(VectorPacket);     { point ES:BX to
vector packet }
        BX := Ofs(VectorPacket);
        Intr(_LASERVECTOR,__IntReg);
        __AddVector := AL;
      end;
    end else __AddVector := 0;
  end;

{-------------------------------------------------------

    __AddOpenShutter function
      add the open shutter command to the background
command queue.  The shutter will be opened by the
background task when it gets to this command in the
command queue.

    returns error code

-------------------------------------------------------}
```

```
function  __AddOpenShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0201;
      Intr(_LASERVECTOR,__IntReg);
      __AddOpenShutter := AL;
    end;
  end;

{-------------------------------------------------------

    __AddCloseShutter function
      add the close shutter command to the background
command queue.  The shutter will be closed by the
background task when it gets to this command in the
command queue.

    returns error code

-------------------------------------------------------}

function __AddCloseShutter: Integer;
  begin
    with __IntReg do begin
      AX := $0202;
      Intr(_LASERVECTOR,__IntReg);
      __AddCloseShutter := AL;
    end;
  end;

{-------------------------------------------------------

    __AddChangeTimebase(NewTicks) function
      add the change time base command to the background
command queue.  The timer that awakens the badckground
task will be reloaded with the count of NewTicks by the
```

```
background task when it gets to this command in the
command queue.

    returns error code

-----------------------------------------------------}

function __AddChangeTimebase(NewTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0203;
      BX := NewTicks;
      Intr(_LASERVECTOR,__IntReg);
      __AddChangeTimebase := AL;
    end;
  end;

{-----------------------------------------------------

    __AddPause(DelayTicks) function
      add the pause command to the background command
queue.  When the background task gets to this command,
it will pause for DelayTicks number of timer ticks.  If
the Background Interrupt Time is 119 (100 us), then a
pause of 1 ms will ensue if DelayTicks is 10.
Afterwards, background activity resumes.

    returns error code

-----------------------------------------------------}

function __AddPause(DelayTicks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0204;
      BX := DelayTicks;
      Intr(_LASERVECTOR,__IntReg);
```

```
      __AddPause := AL;
    end;
  end;

{-------------------------------------------------------

    __ResetCmdQueue function
      clear out the background command queue.  Usually
used as part of an abort procedure.  Best when used just
before a __CloseLaser call.

    returns error code

-------------------------------------------------------

function __ResetCmdQueue: Integer;
  begin
    with __IntReg do begin
      AX := $0300;
      Intr(_LASERVECTOR,__IntReg);
      __ResetCmdQueue := AL;
    end;
  end;

{-------------------------------------------------------

    __GetCmdQueueSize function
      get the background command queue's size in actual
number of bytes.  Number of vectors is variable but can
be estimated by the number returned by this routine /
13.

    returns error code

-------------------------------------------------------}

function __GetCmdQueueSize(var size: Word): Integer;
```

```
  begin
    with __IntReg do begin
      AX := $0400;
      Intr(_LASERVECTOR,__IntReg);
      size := BX;
      if size<0 then size := size + 65536;
      __GetCmdQueueSize := AL;
    end;
  end;

{-------------------------------------------------------

    __GetIntNum(num) function
      place in num the software interrupt number.  Not
really important unless we have a software interrupt
conflict.

    returns error code

-------------------------------------------------------}

function __GetIntNum(var num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0500;
      Intr(_LASERVECTOR,__IntReg);
      num := BL;
      __GetIntNum := AL;
    end;
  end;

{-------------------------------------------------------

    __GetBkgndTime(ticks) function
      place in time the background task interrupt
interval time in actual 1.19 MHz tick counts.  (119 is
100 us, 1190 is 1 ms, etc.)
```

```
    returns error code

-------------------------------------------------------}

function __GetBkgndTime(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0501;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetBkgndTime := AL;
    end;
  end;

{-------------------------------------------------------

    __GetControlWord(ctrl) function
      place in ctrl the Laser Control Word showing the
status of various hardware and software mechanisms.

      word bit  description
      --------  ---------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          0     shutter control, 0=open/1=closed

    returns error code

---   -------------------------------------------------}

function __GetControlWord(var ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0502;
```

```
      Intr(_LASERVECTOR,__IntReg);
      ctrl := BX;
      __GetControlWord := AL;
    end;
  end;

{-------------------------------------------------------

    __AutoShutterModeOn function
      returns the status of the Auto Shutter Mode (true
for on).

-------------------------------------------------------}

function __AutoShutterModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __AutoShutterModeOn := (ctrl and __BITVALUE[0]) > 0;
  end;

{-------------------------------------------------------

    __PosAckModeOn function
      returns the status of the Position Acknowledge
Mode (true for on).

-------------------------------------------------------}

function __PosAckModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
```

```
    __PosAckModeOn := (ctrl and __BITVALUE[1]) > 0;
  end;

{-------------------------------------------------------

    __BlockModeOn function
      returns the status of the Block Mode (true for
on).

-------------------------------------------------------}

function __BlockModeOn: Boolean;
  var
    Er: Integer;
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    __BlockModeOn := (ctrl and __BITVALUE[2]) > 0;
  end;

{-------------------------------------------------------

    __GetStatus(status1,status2) function
      place in status1 and status2 the Laser system's
stati of various non-programmable hardware and software
mechanisms.

      word bit      description
      --------
-------------------------------------------------------
      status1.0     laser system channel open status
      status1.1     laser system software interrupt
established (sic)
      status1.2     laser system background task in idle
state
      status1.3-15  -
```

```
      status2.0     bit 0 of Port C1, X and Y-axis
position acknowledge
      status2.1     bit 1 of Port C1, Z-axis position
acknowledge
      status2.2     bit 2 of Port C1
      status2.3     bit 3 of Port C1
      status2.4-15  -

    returns error code

-------------------------------------------------------}

function __GetStatus(var status1,status2: Word):
Integer;
  begin
    with __IntReg do begin
      AX := $0503;
      Intr(_LASERVECTOR,__IntReg);
      status1 := BX;
      status2 := CX;
      __GetStatus := AL;
    end;
  end;

{-------------------------------------------------------

    __GetOpenStatus function
      tells whether or not the laser system
communications channel has been opened and the timer
tick trap installed.  The function returns a boolean
value of TRUE if the channel is open.

    returns boolean value

-------------------------------------------------------}

function __GetOpenStatus: Boolean;
```

```
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetOpenStatus := (stat1 and
__BITVALUE[0]) > 0
      else __GetOpenStatus := false;
  end;

{-------------------------------------------------

    __GetBkgndIdleStatus function
      tells whether or not the laser system background
task is currently in the idle state where no more laser
activity can be performed without some more foreground-
sourced commands.  The function returns a boolean value
of TRUE if the background task is idle.

    returns boolean value

-------------------------------------------------}

function __GetBkgndIdleStatus: Boolean;
  var
    Er: Integer;
    stat1,stat2: Word;
  begin
    Er := __GetStatus(stat1,stat2);
    if Er=0 then __GetBkgndIdleStatus := (stat1 and
__BITVALUE[2]) > 0
      else __GetBkgndIdleStatus := false;
  end;

{-------------------------------------------------

    __GetTurnOffDelay(ticks) function
```

```
      get the idle state's laser turn-off delay in
interrupt tick counts.

    returns error code

-------------------------------------------------------}

function __GetTurnOffDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0504;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOffDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __GetTurnOnDelay(ticks) function
      get the idle state's laser turn-on delay in
interrupt tick counts.

    returns error code

-------------------------------------------------------}

function __GetTurnOnDelay(var ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0505;
      Intr(_LASERVECTOR,__IntReg);
      ticks := BX;
      __GetTurnOnDelay := AL;
    end;
  end;
```

```
{-------------------------------------------------------

    __GetHomePosition(X,Y) function
      get the idle state's laser home position.

    returns error code

-------------------------------------------------------}

function __GetHomePosition(var X,Y: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0506;
      Intr(_LASERVECTOR,__IntReg);
      X := BX;
      Y := CX;
      __GetHomePosition := AL;
    end;
  end;


{-------------------------------------------------------

    __SetIntNum(num) function
      set the software interrupt number to num.  All
software interrupt call after this point must use this
new interrupt number.  The interrupt vector for the old
number is restored.

    returns error code

-------------------------------------------------------}

function __SetIntNum(num: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0600;
```

```
      BL := num;
      Intr(_LASERVECTOR,__IntReg);
      __SetIntNum := AL;
    end;
  end;

{-------------------------------------------------------

    __SetBkgndTime(ticks) function
      set the background task interrupt interval to
ticks which has the actual number of 1.19 MHz tick
counts.  (119 is 100 us, etc.)

    returns error code

-------------------------------------------------------}

function __SetBkgndTime(ticks: Word): Integer;
  begin
    if ticks<_MINIMUM_TICKS then ticks :=
_MINIMUM_TICKS;
    with __IntReg do begin
      AX := $0601;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetBkgndTime := AL;
    end;
  end;

{-------------------------------------------------------

    __SetControlWord(ctrl) function
      set the Laser Control Word to ctrl.  It is
recommended that most software programs use the
__AlterControlWordBit and the specific mode set routines
which follow.
```

```
      word bit  description
      --------  --------------------------------
          0     Auto-Shutter Mode
          1     Position-Acknowledge Mode
          2     Block Mode
          3     -
          4     shutter control, 0=open/1=closed

    returns error code

-------------------------------------------------------}

function __SetControlWord(ctrl: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0602;
      BX := ctrl;
      Intr(_LASERVECTOR,__IntReg);
      __SetControlWord := AL;
    end;
  end;

{-------------------------------------------------------

    __AlterControlWordBit(op,bit) function
      alter the state of the indicated Control Word bit.
If op is 'S' or '+', the bit is set;  if op is 'C' or
'-', the bit is cleared.

    returns error code

-------------------------------------------------------}

function __AlterControlWordBit(op: Char;bit: Word):
Integer;
  var
    Er: Integer;
```

```
    ctrl: Word;
  begin
    Er := __GetControlWord(ctrl);
    if Er=0 then begin
      if upcase(op) in ['S','+'] then ctrl := ctrl or
__BITVALUE[bit]
        else if upcase(op) in ['C','-'] then
          ctrl := ctrl and (__ALLBITS-__BITVALUE[bit]);
      Er := __SetControlWord(ctrl);
    end;
    __AlterControlWordBit := Er;
  end;

{-------------------------------------------------------

    __SetAutoShutterMode(state) function
      set the state ('S'or '+' to set, 'C' or '-' to
clear) of the Auto-Shutter Mode.

    returns error code

-------------------------------------------------------}

function __SetAutoShutterMode(op: Char): Integer;
  begin
    __SetAutoShutterMode := __AlterControlWordBit(op,0);

  end;

{-------------------------------------------------------

    __SetPosAckMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the Position-Acknowledge Mode.

    returns error code
```

```
-------------------------------------------------------}

function __SetPosAckMode(op: Char): Integer;
  begin
    __SetPosAckMode := __AlterControlWordBit(op,1);
  end;

{-------------------------------------------------------

    __SetBlockMode(state) function
      set the state ('S' or '+' to set, 'C' or '-' to
clear) of the Block Mode where the 07 commands (below)
are active.

    returns error code

-------------------------------------------------------}

function __SetBlockMode(op: Char): Integer;
  begin
    __SetBlockMode := __AlterControlWordBit(op,2);
  end;

{-------------------------------------------------------

    __SetTurnOffDelay(ticks) function
      sets the idle state's laser turn-off delay to
ticks interrupt counts.

    returns error code

-------------------------------------------------------}

function __SetTurnOffDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0604;
```

```
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOffDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __SetTurnOnDelay(ticks) function
      sets the idle state's laser turn-on delay to ticks
interrupt counts.

    returns error code

-------------------------------------------------------}

function __SetTurnOnDelay(ticks: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0605;
      BX := ticks;
      Intr(_LASERVECTOR,__IntReg);
      __SetTurnOnDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __SetHomePosition(X,Y) function
      sets the idle state's laser home position.

    returns error code

-------------------------------------------------------}

function __SetHomePosition(X,Y: Word): Integer;
  begin
```

```
    with __IntReg do begin
      AX := $0606;
      BX := X;
      CX := Y;
      Intr(_LASERVECTOR,__IntReg);
      __SetHomePosition := AL;
    end;
  end;

{-------------------------------------------------------

    __Execute function
      set the background task running.  The background
task will process those commands added to the command
queue from the last Clear Operation up to this __Execute
call.   For the first __Execute, then background task
will start at the beginning of the queue.  New commands
can be added to the command queue while the background
task is running.

      When the background task is finished processing
the current command block, it will automatically shut
down, placing the laser beam at the preset home
position.

      To make programs written for earlier versions
(where the background task free-ran) compatible,
__Execute, __ExecuteClear, and __Clear calls (see below)
should be used wherever appropriate.

    returns error code

-------------------------------------------------------}

function __Execute: Integer;
  begin
    with __IntReg do begin
```

```
      AX := $0700;
      Intr(_LASERVECTOR,__IntReg);
      __Execute := AL;
   end;
  end;

{-------------------------------------------------------

    __ExecuteClear function
      perform the equivalent of the __Execute function
(see above) and a __Clear function (see below) function.
The __ExecuteClear function provides a convenient
shorthand for:

        Er := __Execute;
        Er := __Clear;

    returns error code

-------------------------------------------------------}

function __ExecuteClear: Integer;
  begin
    with __IntReg do begin
      AX := $0701;
      Intr(_LASERVECTOR,__IntReg);
      __ExecuteClear := AL;
    end;
  end;

{-------------------------------------------------------

    __Clear function
      clear the command queue.  Future __Execute and
__ExecuteClear calls will start the background task
processing command queue entries added after the call to
```

```
this routine.  An Execute Operation asked for
immediately after a Clear Operation has no effect.

      If __Clear is not called between consecutive
Execute Operations, then background task will reprocess
the command queue entries for the first Execute
Operation.  Sometimes this is desired, and sometimes it
isn't.

    returns error code

-------------------------------------------------------}

function __Clear: Integer;
  begin
    with __IntReg do begin
      AX := $0702;
      Intr(_LASERVECTOR,__IntReg);
      __Clear := AL;
    end;
  end;

{-------------------------------------------------------

    __SetStepPeriods function
      set the step periods for the multiple passes.
array contains number of step periods to define (1 to
10) and the step period values in 0.82 us (1.19 Mhz)
units.

    returns error code

-------------------------------------------------------}

function __SetStepPeriods(var sp_array:
_StepPeriodsType): Integer;
  begin
```

```
    with __IntReg do begin
      AX := $0800;
      ES := Seg(sp_array);      { point ES:BX to step
periods array }
      BX := Ofs(sp_array);
      Intr(_LASERVECTOR,__IntReg);
      __SetStepPeriods := AL;
    end;
  end;

{-------------------------------------------------------

    __SetRedrawCount function
      set the number of passes for the background task's
redraw (1-8).

    returns error code

-------------------------------------------------------}

function __SetRedrawCount(count: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0801;
      if count<1 then count := 1
        else if count>10 then count := 10;
      BX := count;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawCount := AL;
    end;
  end;

{-------------------------------------------------------

    __SetRedrawDelay function
      set the number of timer ticks the background task
should delay before starting a redraw pass.
```

```
    returns error code

 ------------------------------------------------------}

function __SetRedrawDelay(delay: Word): Integer;
  begin
    with __IntReg do begin
      AX := $0802;
      BX := delay;
      Intr(_LASERVECTOR,__IntReg);
      __SetRedrawDelay := AL;
    end;
  end;

{-------------------------------------------------------

    __Mark function
      marks the current position as the start of the
next redraw.  Usually only necessary at the beginning of
a block.

    returns error code

-------------------------------------------------------}

function __Mark: Integer;
  begin
    with __IntReg do begin
      AX := $0803;
      Intr(_LASERVECTOR,__IntReg);
      __Mark := AL;
    end;
  end;

{-------------------------------------------------------
```

```
    __Redraw function
      causes the background task to perform a multi-pass
redraw operation when it is encountered in the
Background Command Queue.  __Redraw also marks so as to
minimize use of commands.

    returns error code

-------------------------------------------------------}

function __Redraw: Integer;
  begin
    with __IntReg do begin
      AX := $0804;
      Intr(_LASERVECTOR,__IntReg);
      __Redraw := AL;
    end;
  end;

{ end of LASER.INC }^Z
```

```
{-------------------------------------------------------

                          GRAPH Include File

-------------------------------------------------------}

{$F+} procedure MainExit; {$F-}
  begin
    {$I-} Close(SliceFile); Close(TriangleFile); {$I+}
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrScr;
    ExitProc:=ExitSave;
  end;

procedure CursorOff;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
      AH := 1;
      CH := CH or $20;
      Intr($10,regs);
   end;
end;

procedure CursorOn;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
      AH := 1;
      CH := CH and $DF;
```

```
      Intr($10,regs);
    end;
end;


{*******************************************************
procedure ConvertRtoS(r,s : PointerType);
  { Converts a Real type (6 byte) number to a Single (4
byte)
var
  e : Byte;
begin
  e:=r^[0];
  if e<>0 then Dec(e,2);
  s^[3]:=(r^[5] and $80) or ((e shr 1) and $7F);
  s^[2]:=((e shl 7) or (r^[5] and $7F)) and $FF;
  s^[1]:=r^[4];
  s^[0]:=r^[3];
end;
*******************************************************}

procedure ProgramError(ErrorLine : String);
  { Print Program Error and exits program }
  begin
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrScr;
    writeln('*** SLAGRAPH ERROR: ',ErrorLine,' ***');
    ch:=ReadKey;
    Halt(1);
  end;

procedure ReAllocate(ch : Char;Alloc : LongInt);
  { Allocates the array space to sugessted number }
  var
    MemoryAvail : LongInt;
```

```
 begin
   { Free Heap }
   Release(HeapTop);
   SliceFilename   := '';
   TriangleFilename:= '';
   MemoryAvail := MemAvail - 32768;

   if ch='S' then begin
     { Don't allow more than machine can handle }
     if Alloc>MAX_LAYERS then Total_Layers:=MAX_LAYERS
     else Total_Layers:=Alloc;
     { Make sure you can allocate that many alone. If
not, set maximum }
     if Total_Layers*LAYER_SIZE>MemoryAvail then
Total_Layers:=Round(MemoryAvail/LAYER_SIZE);
     { Given number of Layers, fill rest of memory with
Triangles }
     Total_Triangles:=Round((MemoryAvail-Total_Layers*
LAYER_SIZE)/TRIANGLE_SIZE);
     if Total_Triangles>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES;
   end else begin
     { Don't allow more than machine can handle }
     if Alloc>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES
     else Total_Triangles:=Alloc;
     { Make sure you can allocate that many alone. If
not, set maximum }
     if Total_Triangles*TRIANGLE_SIZE>MemoryAvail then
Total_Triangles:=Round(MemoryAvail/TRIANGLE_SIZE);
     { Given number of Triangles, fill rest of memory
with Layers }
     Total_Layers:=Round((MemoryAvail-Total_Triangles*
TRIANGLE_SIZE)/LAYER_SIZE);
     if Total_Layers>MAX_LAYERS then
Total_Layers:=MAX_LAYERS;
   end;
```

```
    GetMem(SliceLayerValue,Total_Layers*2);
    GetMem(SliceLayerPtr,Total_Layers*4);
    GetMem(SliceLayerBlocks,Total_Layers*2);
    GetMem(SliceLayerXMin,Total_Layers*2);
    GetMem(SliceLayerXMax,Total_Layers*2);
    GetMem(SliceLayerYMin,Total_Layers*2);
    GetMem(SliceLayerYMax,Total_Layers*2);
    GetMem(TrianglePtr,Total_Triangles*4);
    GetMem(TriangleAxisValue,Total_Triangles*6);
    Dec(Total_Triangles);
    Dec(Total_Layers);
  end;


procedure OutTextLn(aline:String);
  { "writeln" for graphics }
  begin
    OutText(aline);
    MoveTo(1,GetY+TextHeight(' ')+2);
  end;

procedure GrTextXY(aline:String;x,y : Byte);
  { writes text at XY char on graphic screen }
  begin
    OutTextXY((x-1)*(TextWidth('
')+2),(y-1)*(TextHeight(' ')+2),aline);
  end;

procedure ConvertStoR(s,r : PointerType);
  { Converts a Single type (4 byte) number to a Real (6
byte) }
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
```

```
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;

procedure ToUpper(var aline : String);
  { Converts a string to upper case }
  var
    i : Integer;
  begin
    for i:=1 to Length(aline) do aline[i] :=
UpCase(aline[i]);
  end;

procedure Center(str : String;linenum : Byte);
  { Centers text on screen line }
  begin
    GotoXY(1,linenum);
    ClrEol;
    if Length(str)>79 then GotoXY(1,linenum)
    else GotoXY(40-(Length(str) div 2),linenum);
    write(str);
  end;

procedure WaitKey;
  { Waits for a key press while prompting }
  var
    ch : Char;
  begin
    Center('<Press any key to continue>',25);
    ch:=ReadKey;
  end;

procedure PrintHeader;
  { Prints general 3D header at top of screen }
  begin
    TextColor(White);
    if Not MonoDisplay then TextBackground(Blue)
```

```
    else TextBackground(Black);
    ClrScr;
    Center('3D Systems Graphics Display Program (Version
'+VERSION+')',1);
    Center('Copyright (C) 1988 3D Systems Inc',2);
    SelectTxtColor(LightGreen);
  end;

procedure PrintSliceHeader;
  { Print Slice Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Slice File Control Functions',3);
    TextColor(White);
  end;

procedure PrintTriHeader;
  { Print Triangle Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Triangle File Control Functions',3);
    TextColor(White);
  end;

function GetFilename(var FileName : String; sel :
Integer) : Boolean;
  { Displays file with SLI or STL extentions and allows
user to select
     a file to use. }
  var
    DirRec          : SearchRec;
    CurrFileName    : String;
    Extn            : String;
    row,col         : Integer;
    count           : Integer;
```

```
begin
  DisplayMode := 'W';
  FileSorted  := false;
  FileError:=FALSE;
  if sel = 1 then begin
    Extn := '.STL';
    PrintTriHeader;
    GotoXY(3,6);
    write('Triangle');
  end else begin
    Extn := '.SLI';
    PrintSliceHeader;
    GotoXY(3,6);
    write('Slice');
  end;
  { Display the entries }
  write(' Files in current directory:');
  row := 0;
  col := 0;
  count := 0;
  FindFirst('*'+Extn,AnyFile,DirRec);
  while (DosError = 0) and (count < 75) do begin
    GotoXY(3+col*15,8+row);
    write(Copy(DirRec.name,1,Pos('.',DirRec.name)-1));

    col := col + 1;
    if col = 5 then begin
      col := 0;
      row := row + 1;
    end;
    FindNext(DirRec);
  end;
  { Read the filename }
  repeat
    GotoXY(1,24);
    ClrEol;
    write('   Enter File Name: ');
```

```
      Readln(CurrFilename);
      ToUpper(CurrFilename);
      DosError:=0;
      if CurrFilename <> '' then begin
        { Find in directory }
        if Pos('.',CurrFilename) = 0 then CurrFilename
:= CurrFilename + Extn;
        if Pos('*',CurrFilename)=0 then begin
          FindFirst(CurrFilename,AnyFile,DirRec);
          if DosError <> 0 then Center('****
"'+CurrFilename+'" File Not Found ****',25);
        end else begin
          DosError:=1;
          Center('*** No Wild cards allowed ***',25);
        end;
        if DosError<>0 then CurrFilename:='';
      end;
    until (DosError = 0);
    if CurrFilename = '' then GetFilename:=FALSE
    else begin
      FileName := CurrFilename;
      GetFilename:=TRUE;
    end;
  end;

procedure ASCError(FileName : String);
  { Prints ASC file error }
  begin
    Str(StartLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
byte offset '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
  end;

procedure ASCErrorLn(FileName : String);
  { Prints ASC file error }
```

```
  begin
    Str(CurrLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
line '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
  end;

procedure CheckBuffer(var InFile : File);
  { Makes sure the input buffer has enogh data in it.
     If not, it reads more in. }
  var
    NumRead : Word;
  begin
    if InBufPtr > InBufEnd-160 then begin
      { Reload Buffer }
      if InBufPtr > InBufEnd then InBufEnd := 0
{ Buffer Empty }
      else begin
        { Buffer almost empty }
         InBufEnd := InBufEnd - InBufPtr + 1;
         Move(InBuffer[InBufPtr],InBuffer[0],InBufEnd);
      end;
      { Fill rest of buffer }
      if Not Eof(InFile) then BlockRead(InFile,InBuffer
[InBufEnd],BUF_SIZE-InBufEnd, NumRead)
      else NumRead:=0;
      { Set buffer pointers }
      if InBufEnd+NumRead=0 then EndOfFile:=TRUE
      else Inc(InBufEnd,NumRead - 1);
      InBufPtr := 0;
    end;
  end;

procedure ResetFile(var InFile : File);
  { Resets input file to top of file }
  begin
```

```
    Reset(InFile,1);
    Seek(InFile,0);
    CurrFilePos:=0;
    CurrLinePos:=0;
    InBufPtr := BUF_SIZE+1;
    InBufEnd := BUF_SIZE;
    EndOfFile:=FALSE;
    FileError:=FALSE;
  end;

procedure PositionFile(var InFile : File;charnum :
LongInt);
  { Positions input buffer to specific character of
file.  If not in buffer reads from file }
  begin
    if (charnum>=CurrFilePos-InBufPtr) AND
(charnum<=CurrFilePos-InBufPtr+InBufEnd) then
      InBufPtr:=InBufPtr+charnum-CurrFilePos
    else begin
      Seek(InFile,charnum);
      InBufPtr := BUF_SIZE+1;
      InBufEnd := BUF_SIZE;
      EndOfFile:=FALSE;
    end;
    CurrFilePos:=charnum;
    CheckBuffer(InFile);
  end;

procedure ReadLine(var InFile : File);
  { Reads a line from input file into a string }
  var
    EOFPos,CRPos,i        : Integer;
  begin
    InputLine:='';
    if Not (FileError or EndOfFile) then repeat
      CheckBuffer(InFile);
      { Search for CR }
```

```
      if Not EndOfFile then begin
        if InBufEnd=0 then i:=0
        else i := InBufEnd-InBufPtr+1;
        if i > 255 then i := 255;
        InputLine[0] := Char (i);
        Move(InBuffer[InBufPtr],InputLine[1],i);
        CRPos := Pos(CR,InputLine)-1;
        if CRPos < 0 then CRPos := i;
        InputLine[0] := Char (CRPos);
        EOFPos := Pos(EOF_CHAR,InputLine)-1;
        if EOFPos>0 then InputLine[0] := Char (EOFPos);
        StartLinePos:=CurrFilePos;
        PositionFile(InFile,CurrFilePos+CRPos+1);
        if InBuffer[InBufPtr] = LF then PositionFile
(InFile,CurrFilePos+1);
        Inc(CurrLinePos);
        ToUpper(InputLine);
        ErrLine:=InputLine;
      end else InputLine := '';
    until EndOfFile OR (InputLine <> '');
    EndOfFile := EndOfFile and (InputLine='');
  end;

function ParseLine(var aline : String) : String;
  { Parses first word from string and truncs off from
original string }
  var
    parsed : String;
    i,j    : Integer;
  begin
    i:=1;
    while (i<=Length(aline)) AND (aline[i]=' ') do
Inc(i);
    if i>Length(aline) then begin
      aline:='';
      parsed:='';
    end else begin
```

```
        j:=i;
        while (j<=Length(aline)) AND (aline[j]<>' ') do
Inc(j);
        Move(aline[i],Parsed[1],j-i);
        parsed[0]:=Chr (j-i);
        if j>=Length(aline) then aline:=''
        else begin
          Move(aline[j+1],aline[1],Length(aline)-j);
          aline[0]:=Char (Length(aline)-j);
        end;
      end;
      ParseLine:=parsed;
    end;

^Z
```

```
{-------------------------------------------------------

                          NEWVIEW Include File

-------------------------------------------------------}

function RealVal(line: string;var code: Integer):Real;
var
  i,code2 : integer;
  num     : real;
begin
  if line[1]='.' then line := '0' + line;
  if (line[1]='-') and (line[2]='.') then begin
    line := '-0' + line;
    for i:= 4 to length(line) do line[i-1]:=line[i];
    line[length(line)] := ' ';
  end;
  val(line,num,code);
  if code <> 0 then begin
    val(copy(line,1,code-1),num,code2);
    code := code2;
  end;
  RealVal := num;
end;
{$F+} procedure MainExit; {$F-}
  begin
    {$I-} Close(SliceFile); Close(TriangleFile); {$I+}
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrSc
    ExitP  :=ExitSave;
  end;

procedure SelectTxtColor(select : Integer);
  { Sets text color requested or White for non-color
displays }
```

```
  begin
    if MonoDisplay then TextColor(White)
    else TextColor(select);
  end;


procedure SelectGraphColor(select : Integer);
  { Sets graphic color.  If black and white, uses dashed
lines, etc. }
  var
    LineType : Integer;
  begin
    if MonoDisplay then begin
      SetColor(White);
      case select of
         0 : SetColor(Black);
         1 : LineType:=SolidLn;
         2 : LineType:=CenterLn;
         4 : begin
               LineType:=CenterLn;
               SetColor(LightGray);
             end;
         9 : begin
               LineType:=SolidLn;
               SetColor(LightGray);
             end;
        10 : LineType:=DashedLn;
        12 : begin
               LineType:=DashedLn;
               SetColor(LightGray);
             end;
        ELSE LineType:=CenterLn;
      end;
      SetLineStyle(LineType,0,NormWidth);
    end else SetColor(select);
  end;
```

```
procedure CursorOff;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
      AH := 1;
      CH := CH or $20;
      Intr($10,regs);
   end;
end;

procedure CursorOn;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
      AH := 1;
      CH := CH and $DF;
      Intr($10,regs);
   end;
end;


{*******************************************************
procedure ConvertRtoS(r,s : PointerType);
  { Converts a Real type (6 byte) number to a Single (4
byte)
var
  e : Byte
begin
  e:=r^[0];
  if e<>0 then Dec(e,2);
  s^[3]:=(r^[5] and $80) or ((e shr 1) and $7F);
  s^[2]:=((e shl 7) or (r^[5] and $7F)) and $FF;
  s^[1]:=r^[4];
```

```
  s^[0]:=r^[3];
end;
*****************************************************}

procedure ProgramError(ErrorLine : String);
  { Print Program Error and exits program }
  begin
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrScr;
    writeln('*** SLAGRAPH ERROR: ',ErrorLine,' ***');
    ch:=ReadKey;
    Halt(1);
  end;

procedure ReAllocate(ch : Char;Alloc : LongInt);
  { Allocates the array space to sugessted number }
  var
    MemoryAvail : LongInt;
  begin
    { Free Heap }
    Release(HeapTop);
    SliceFilename   := '';
    TriangleFilename:= '';
    MemoryAvail := MemAvail - 32768;

    if ch='S' then begin
      { Don't allow more than machine can handle }
      if Alloc>MAX_LAYERS then Total_Layers:=MAX_LAYERS
      else Total_Layers:=Alloc;
      { Make sure you can allocate that many alone. If
not, set maximum }
      if Total_Layers*LAYER_SIZE>MemoryAvail then
Total_Layers:=Round(MemoryAvail/LAYER_SIZE);
      { Given number of Layers, fill rest of memory with
Triangles }
```

```
      Total_Triangles:=Round((MemoryAvail-Total_Layers*
LAYER_SIZE)/TRIANGLE_SIZE);
      if Total_Triangles>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES;
    end else begin
      { Don't allow more than machine can handle }
      if Alloc>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES
      else Total_Triangles:=Alloc;
      { Make sure you can allocate that many alone.  If
not, set maximum }
      if Total_Triangles*TRIANGLE_SIZE>MemoryAvail then
Total_Triangles:=Round(MemoryAvail/TRIANGLE_SIZE);
      { Given number of Triangles, fill rest of memory
with Layers }
      Total_Layers:=Round((MemoryAvail-Total_Triangles*
TRIANGLE_SIZE)/LAYER_SIZE);
      if Total_Layers>MAX_LAYERS then Total_Layers:=
MAX_LAYERS;
    end;
    GetMem(SliceLayerValue,Total_Layers*2);
    GetMem(SliceLayerPtr,Total_Layers*4);
    GetMem(SliceLayerBlocks,Total_Layers*2);
    GetMem(SliceLayerXMin,Total_Layers*2);
    GetMem(SliceLayerXMax,Total_Layers*2);
    GetMem(SliceLayerYMin,Total_Layers*2);
    GetMem(SliceLayerYMax,Total_Layers*2);
    GetMem(TrianglePtr,Total_Triangles*4);
    GetMem(TriangleAxisValue,Total_Triangles*6);
    Dec(Total_Triangles);
    Dec(Total_Layers);
  end;


procedure OutTextLn(aline:String);
  { "writeln" for graphics }
  begin
```

```
    OutText(aline);
    MoveTo(1,GetY+TextHeight(' ')+2);
  end;

procedure GrTextXY(aline:String;x,y : Byte);
  { writes text at XY char on graphic screen }
  begin
    OutTextXY((x-1)*(TextWidth('
')+2),(y-1)*(TextHeight(' ')+2),aline);
  end;

procedure ConvertStoR(s,r : PointerType);
  { Converts a Single type (4 byte) number to a Real (6
byte) }
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;

procedure ToUpper(var aline : String);
  { Converts a string to upper case }
  var
    i : Integer;
  begin
    for i:=1 to Length(aline) do aline[i] :=
UpCase(aline[i]);
  end;

procedure Center(str : String;linenum : Byte);
  { Centers text on screen line }
  begin
    GotoXY(1,linenum);
```

```
    ClrEol;
    if Length(str)>79 then GotoXY(1,linenum)
    else GotoXY(40-(Length(str) div 2),linenum);
    write(str);
  end;

procedure WaitKey;
  { Waits for a key press while prompting }
  var
    ch : Char;
  begin
    Center('<Press any key to continue>',25);
    ch:=ReadKey;
  end;

procedure PrintHeader;
  { Prints general 3D header at top of screen }
  begin
    TextColor(White);
    if Not MonoDisplay then TextBackground(Blue)
    else TextBackground(Black);
    ClrScr;
    Center('3D Systems Graphics Display Program (Version
'+VERSION+')',1);
    Center('Copyright (C) 1988 3D Systems Inc',2);
    SelectTxtColor(LightGreen);
  end;

procedure PrintSliceHeader;
  { Print Slice Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Slice File Control Functions',3);
    TextColor(White);
  end;
```

```
procedure PrintTriHeader;
  { Print Triangle Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Triangle File Control Functions',3);
    TextColor(White);
  end;

function GetFilename(var FileName : String; sel :
Integer) : Boolean;
  { Displays file with SLI or STL extentions and allows
user to select
     a file to use. }
  var
    DirRec          : SearchRec;
    CurrFileName    : String;
    Extn            : String;
    row,col         : Integer;
    count           : Integer;
  begin
    DisplayMode := 'W';
    WSorted  := true; { because when we get the file it
will be wireframe }
    SSorted  := false;
    FileError:=FALSE;
    if sel = 1 then begin
      Extn := '.STL';
      PrintTriHeader;
      GotoXY(3,6);
      write('Triangle');
    end else begin
      Extn := '.SLI';
      PrintSliceHeader;
      GotoXY(3,6);
      write('Slice');
    end;
```

```
      { Display the entries }
      write(' Files in current directory:');
      row := 0;
      col := 0;
      count := 0;
      FindFirst('*'+Extn,AnyFile,DirRec);
      while (DosError = 0) and (count < 75) do begin
        GotoXY(3+col*15,8+row);
        write(Copy(DirRec.name,1,Pos('.',DirRec.name)-1));
        col := col + 1;
        if col = 5 then begin
          col := 0;
          row := row + 1;
        end;
        FindNext(DirRec);
      end;
      { Read the filename }
      repeat
        GotoXY(1,24);
        ClrEol;
        write('   Enter File Name: ');
        Readln(CurrFilename);
        ToUpper(CurrFilename);
        DosError:=0;
        if CurrFilename <> '' then begin
          { Find in directory }
          if Pos('.',CurrFilename) = 0 then CurrFilename
:= CurrFilename + Extn;
          if Pos('*',CurrFilename)=0 then begin
            FindFirst(CurrFilename,AnyFile,DirRec);
            if DosError <> 0 then Center('****
"'+CurrFilename+'" File Not Found ****',25);
          end else begin
            DosError:=1;
            Center('*** No Wild cards allowed ***',25);
          end;
          if DosError<>0 then CurrFilename:='';
```

```
      end;
    until (DosError = 0);
    if CurrFilename = '' then GetFilename:=FALSE
    else begin
      FileName := CurrFilename;
      GetFilename:=TRUE;
    end;
  end;

procedure ASCError(FileName : String);
  { Prints ASC file error }
  begin
    Str(StartLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
byte offset '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
  end;

procedure ASCErrorLn(FileName : String);
  { Prints ASC file error }
  begin
    Str(CurrLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
line '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
  end;

procedure CheckBuffer(var InFile : File);
  { Makes sure the input buffer has enogh data in it.
     If not, it reads more in. }
  var
    NumRead : Word;
  begin
    if InBufPtr > InBufEnd-160 then begin
      { Reload Buffer }
```

```
      if InBufPtr > InBufEnd then InBufEnd := 0  {
Buffer Empty }
      else begin
        { Buffer almost empty }
         InBufEnd := InBufEnd - InBufPtr + 1;
         Move(InBuffer[InBufPtr],InBuffer[0],InBufEnd);
      end;
      { Fill rest of buffer }
      if Not Eof(InFile) then
BlockRead(InFile,InBuffer[InBufEnd],BUF_SIZE-InBufEnd,Nu
mRead)
      else NumRead:=0;
      { Set buffer pointers }
      if InBufEnd+NumRead=0 then EndOfFile:=TRUE
      else Inc(InBufEnd,NumRead - 1);
      InBufPtr := 0;
    end;
  end;

procedure ResetFile(var InFile : File);
  { Resets input file to top of file }
  begin
    Reset(InFile,1);
    Seek(InFile,0);
    CurrFilePos:=0;
    CurrLinePos:=0;
    InBufPtr := BUF_SIZE+1;
    InBufEnd := BUF_SIZE;
    EndOfFile:=FALSE;
    FileError:=FALSE;
  end;

procedure PositionFile(var InFile : File;charnum :
LongInt);
  { Positions input buffer to specific character of
file.  If not in buffer reads from file }
  begin
```

```
    if (charnum>=CurrFilePos-InBufPtr) AND
(charnum<=CurrFilePos-InBufPtr+InBufEnd) then
      InBufPtr:=InBufPtr+charnum-CurrFilePos
    else begin
      Seek(InFile,charnum);
      InBufPtr := BUF_SIZE+1;
      InBufEnd := BUF_SIZE;
      EndOfFile:=FALSE;
    end;
    CurrFilePos:=charnum;
    CheckBuffer(InFile);
  end;

procedure ReadLine(var InFile : File);
  { Reads a line from input file into a string }
  var
    EOFPos,CRPos,i        : Integer;
  begin
    InputLine:='';
    if Not (FileError or EndOfFile) then repeat
      CheckBuffer(InFile);
      { Search for CR }
      if Not EndOfFile then begin
        if InBufEnd=0 then i:=0
        else i := InBufEnd-InBufPtr+1;
        if i > 255 then i := 255;
        InputLine[0] := Char (i);
        Move(InBuffer[InBufPtr],InputLine[1],i);
        CRPos := Pos(CR,InputLine)-1;
        if CRPos < 0 then CRPos := i;
        InputLine[0] := Char (CRPos);
        EOFPos := Pos(EOF_CHAR,InputLine)-1;
        if EOFPos>0 then InputLine[0] := Char (EOFPos);
        StartLinePos:=CurrFilePos;
        PositionFile(InFile,CurrFilePos+CRPos+1);
        if InBuffer[InBufPtr] = LF then
PositionFile(InFile,CurrFilePos+1);
```

```
        Inc(CurrLinePos);
        ToUpper(InputLine);
        ErrLine:=InputLine;
      end else InputLine := '';
    until EndOfFile OR (InputLine <> '');
    EndOfFile := EndOfFile and (InputLine='');
  end;

function ParseLine(var aline : String) : String;
  { Parses first word from string and truncs off from
original string }
  var
    parsed : String;
    i,j    : Integer;
  begin
    i:=1;
    while (i<=Length(aline)) AND (aline[i]=' ') do
Inc(i);
    if i>Length(aline) then begin
      aline:='';
      parsed:='';
    end else begin
      j:=i;
      while (j<=Length(aline)) AND (aline[j]<>' ') do
Inc(j);
      Move(aline[i],Parsed[1],j-i);
      parsed[0]:=Chr (j-i);
      if j>=Length(aline) then aline:=''
      else begin
        Move(aline[j+1],aline[1],Length(aline)-j);
        aline[0]:=Char (Length(aline)-j);
      end;
    end;
    ParseLine:=parsed;
  end;

procedure Switch(First,Second : LongInt) ;
```

```
begin
  TempReal := TriangleAxisValue^[First];
  TriangleAxisValue^[First]:=TriangleAxisValue^[Second];
  TriangleAxisValue^[Second]:=TempReal;
  TempLong := TrianglePtr^[First];
  TrianglePtr^[First]:=TrianglePtr^[Second];
  TrianglePtr^[Second]:=TempLong;
end;


procedure QuickSort(Left,Right : Word);
  var
    LeftIndex,RightIndex : LongInt;
    KeyValue : Real;
begin
    LeftIndex  := Left ;
    RightIndex := Right ;
    KeyValue := TriangleAxisValue^[(Left+Right) DIV 2] ;
    repeat
      while TriangleAxisValue^[LeftIndex]<KeyValue do
Inc(LeftIndex);
      while KeyValue<TriangleAxisValue^[RightIndex] do
Dec(RightIndex);
      if LeftIndex <= RightIndex then begin
        Switch(LeftIndex, RightIndex);
        Inc(LeftIndex);
        Dec(RightIndex);
      end;
    until LeftIndex > RightIndex ;
    if Left<RightIndex then QuickSort(Left, RightIndex);
    if LeftIndex<Right then QuickSort(LeftIndex, Right);
  end;

procedure SetMinMax(var num,min,max : Word);
  { Updates Min and Max values with given data }
  begin
    if num>max then max:=num;
```

```
    if num<min then min:=num;
  end;

function GraphDriverName(DriverNum : Integer) : String;
  begin
    Case DriverNum of
      1 : GraphDriverName:='CGA';
      2 : GraphDriverName:='MCGA';
      3 : GraphDriverName:='EGA';
      4 : GraphDriverName:='EGA 64';
      5 : GraphDriverName:='EGA Monochrome';
      6 : GraphDriverName:='Reserved';
      7 : GraphDriverName:='Hercules Monochrome';
      8 : GraphDriverName:='ATT400';
      9 : GraphDriverName:='VGA';
      10: GraphDriverName:='PC3270';
      else begin
        Str(DriverNum,TempStr);
        GraphDriverName:='Unknown Type '+TempStr;
      end;
    end;
  end;


procedure SetGraph;
  { Finds what type of video hardware is installed and
picks best graphics screen.  Sets screen definition
information. }
  var
    aspectx,aspecty : Word;
  begin
    DetectGraph(GraphDriver,GraphMode);
    if GraphDriver < 0 then ProgramError('Graphics -
'+GraphErrorMsg(GraphDriver));
    if GraphDriver > 5 then ProgramError('Graphics -
'+GraphDriverName(GraphDriver)+' graphics card not
supported');
```

```
    if GraphDriver <=2 then rc:=RegisterBGIdriver
(@CGADriverProc)
    else rc:=RegisterBGIdriver(@EGAVGADriverProc);
    if rc<0 then ProgramError('Graphics -
'+GraphDriverName(GraphDriver)+':
'+GraphErrorMsg(GraphResult));
    GetModeRange(GraphDriver,TempInt,GraphMode);
    if GraphMode>3 then GraphMode:=3; { for CGA }
    InitGraph(GraphDriver, GraphMode, 'c:\tp');
    rc := GraphResult;
    if rc <> GrOk then ProgramError('Graphics -
'+GraphErrorMsg(rc));
    if (GraphDriver<5) or (GraphDriver=8) then
MonoDisplay:=FALSE
    else MonoDisplay:=TRUE;
    GetAspectRatio(aspectx,aspecty);
    AspectRatio := aspectx/aspecty;
    SizeX := GetMaxX + 1;
    SizeY := GetMaxY + 1;
    GraphCenterX:= Round(SizeX/2);
    GraphCenterY:= Round(SizeY/2);
  end;

procedure SetupGraphicW(MinXW,MaxXW,MinYW,MaxYW : Word);
  { Sets up graphic vars using Integer(Word) proposed
ranges }
  var
    scalex : Real;
    MinX,MaxX,MinY,MaxY : Real;
  begin                {here}
    MinX := MinXW;
    MaxX := MaxXW;
    MinY := MinYW;
    MaxY := MaxYW;
    GraphScale:=(SizeX-2)/((MaxX-MinX)/AspectRatio);
    if (MaxY-MinY)*GraphScale>SizeY then
GraphScale:=(SizeY-4)/(MaxY-MinY);
```

```
    GrOffsetX:=MinX+(MaxX-MinX)/2;
    GrOffsetY:=MinY+(MaxY-MinY)/2;
    SetGraphMode(GraphMode);
    SetColor(White);
    SetBkColor(Black);
    SetLineStyle(SolidLn,0,NormWidth);
  end;

procedure SetupGraphicR(MinX,MaxX,MinY,MaxY,MinZ,MaxZ :
Real);
  { Sets up graphic vars using Real values and assuming
rotations }
  var
    RangeX,RangeY,RangeZ : Real;
  begin
    RangeX:=(MaxX-MinX);
    RangeY:=(MaxY-MinY);
    RangeZ:=(MaxZ-MinZ);
    GraphScale :=
SizeY/Sqrt(Sqr(RangeX)+Sqr(RangeY)+Sqr(RangeZ));
    GrOffsetX:=MinX+RangeX/2;
    GrOffsetY:=MinY+RangeY/2;
    GrOffsetZ:=MinZ+RangeZ/2;
    SetGraphMode(GraphMode);
    SetColor(White);
    SetBkColor(Black);
    SetLineStyle(SolidLn,0,NormWidth);
  end;

procedure DrawLine(x1,y1,x2,y2 : Word);
  { Draws a line on graphics screen (no rotation) }
  var
    rx1,ry1,rx2,ry2 : Real;
  begin
    rx1:=x1;
    ry1:=y1;
    rx2:=x2;
```

```
    ry2:=y2;
    x1:=GraphCenterX+Round((rx1-GrOffsetX)*GraphScale/
AspectRatio);
    x2:=GraphCenterX+Round((rx2-GrOffsetX)*GraphScale/
AspectRatio);
    y1:=GraphCenterY-Round((ry1-GrOffsetY)*GraphScale);
    y2:=GraphCenterY-Round((ry2-GrOffsetY)*GraphScale);
    Line(x1,y1,x2,y2);
  end;

procedure Vertex3Dzoom(x,y,z : Real;var new_x,new_z :
Integer);
  { Maps a 3D point into 2D graphics screen }
  begin
    x:=-(x-GrOffsetX)*zoom;
    y:=(GrOffsetY-y)*zoom;
    z:=(z-GrOffsetZ)*zoom;
    new_x := Round(HoriOffset*zoom) +
    (GraphCenterX + Round(GraphScale*(x*Gr3DMat[0,0]+y*
Gr3DMat[1,0]+z*Gr3DMat[2,0])/AspectRatio));

    {  Have Y mapping just for reference.
    new_y := (GraphCenterY - Round(GraphScale*(x*Gr3DMat
[0,1]+y*Gr3DMat[1,1]+z*Gr3DMat[2,1])));
    }

    new_z := Round(-VertOffset*zoom)+
    (GraphCenterY - Round(GraphScale*(x*Gr3DMat[0,2]+y*
Gr3DMat[1,2]+z*Gr3DMat[2,2])));
  end;

procedure Vertex3D(x,y,z : Real;var new_x,new_z :
Integer);
  { Maps a 3D point into 2D graphics screen }
  begin
    x:=-(x-GrOffsetX);
    y:=(GrOffsetY-y);
```

```
    z:=z-GrOffsetZ;
    new_x := (GraphCenterX +
Round(GraphScale*(x*Gr3DMat[0,0]+y*Gr3DMat[1,0]+z*
Gr3DMat[2,0])/AspectRatio));
   {  Have Y mapping just for reference.
    new_y := (GraphCenterY -
Round(GraphScale*(x*Gr3DMat[0,1]+y*Gr3DMat[1,1]+z*
Gr3DMat[2,1])));
    }
    new_z := (GraphCenterY -
Round(GraphScale*(x*Gr3DMat[0,2]+y*Gr3DMat[1,2]+z*
Gr3DMat[2,2])));
  end;

procedure Compute3D(x,y,z : Real;var new_x,new_z :
Integer);
  { Transforms 3D into 2D with no part scaling or
offsetting }
  begin
    new_x :=
Round((-x*Gr3DMat[0,0]-y*Gr3DMat[1,0]+z*Gr3DMat[2,0])/
AspectRatio);
    new_z := Round(-x*Gr3DMat[0,2]-y*Gr3DMat[1,2]+z*
Gr3DMat[2,2]);
  end;

procedure DrawAxis;
  { Draws Axis in courner of screen }
  var
    coorx,coory : Integer;
    origx,origy : Integer;
  begin
    origx:=SizeX-35;
    origy:=35;
    SelectGraphColor(4);
    Compute3D(20,0,0,coorx,coory);
```

```
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    SelectGraphColor(12);
    Compute3D(0,20,0,coorx,coory);
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    SelectGraphColor(1);
    Compute3D(0,0,20,coorx,coory);
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    Compute3D(20,0,0,coorx,coory);
    SelectGraphColor(white);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'X');
    Compute3D(0,20,0,coorx,coory);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'Y');
    Compute3D(0,0,20,coorx,coory);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'Z');
 end;

procedure DrawPartLine(x1,y1,z1,x2,y2,z2 : Real);
  { Takes 2 3D points and draws line between them }
  var
    coorx1,coory1,coorx2,coory2 : Integer;
  begin
    Vertex3Dzoom(x1,y1,z1,coorx1,coory1);
    Vertex3Dzoom(x2,y2,z2,coorx2,coory2);
    Line(coorx1,coory1,coorx2,coory2);
  end;

procedure Init3D;
  { Sets up 3D transformation matrix for mapping 3D to
2D }
  var
    txy,txz,tyz : Real;
    h2,h1       : Integer;
  begin
    h2  :=  TriangleAngle[2] + DY;   { Vertical Axis }
```

```
    h1  :=  TriangleAngle[1] + DX;   { Horizontal Axis }

    if h2>=360 then h2 := h2-360;
    if h1>=360 then h1 := h1-360;
    txy :=  -TriangleAngle[3]*0.0175; { Into Screen}
    txz :=  h2*0.0175; { Vertical Axis }
    tyz :=  h1*0.0175; { Horizontal Axis }

    Gr3DMat[0,0]:= Cos(txz)*Cos(txy);
    Gr3DMat[1,0]:=
Cos(tyz)*Sin(txy)-Sin(tyz)*Sin(txz)*Cos(txy);

Gr3DMat[2,0]:=-Sin(tyz)*Sin(txy)-Cos(tyz)*Sin(txz)*Cos(t
xy);
    Gr3DMat[0,1]:=-Cos(txz)*Sin(txy);
    Gr3DMat[1,1]:=
Cos(tyz)*Cos(txy)+Sin(tyz)*Sin(txz)*Sin(txy);

Gr3DMat[2,1]:=-Sin(tyz)*Cos(txy)+Cos(tyz)*Sin(txz)*Sin(t
xy);
    Gr3DMat[0,2]:= Sin(txz);
    Gr3DMat[1,2]:= Sin(tyz)*Cos(txz);
    Gr3DMat[2,2]:= Cos(tyz)*Cos(txz);
{
writeln(gftest,'mat',Gr3dmat[0,1],'/',gr3dmat[1,1],'/',g
r3dmat[2,1]);}
  end;


{-----------------------------------------------------
  |     Start of Main Triangle Routines          |

-----------------------------------------------------}

function FindTriangleLevel(value : Real;highlo : Byte) :
Word;
```

```
  { Binary search for value and then file lowest or
highest common index }
  var
    low,mid,high,index  :  Word;
  begin
    low:=0;
    high:=TriangleTotal;
    index:=MAX_WORD;
    repeat
      mid:=(low+high) div 2;
      if value<TriangleAxisValue^[mid] then high:=mid
      else if value>TriangleAxisValue^[mid] then
low:=mid
           else index:=mid;
    until (high-low=1) or (index<MAX_WORD);
    if highlo=1 then begin
      if index=MAX_WORD then index:=high;
      while (index>0) and
(TriangleAxisValue^[index-1]=TriangleAxisValue^[index])
do Dec(index)
    end else begin
      if index=MAX_WORD then index:=low;
      while (index<TriangleTotal) and
(TriangleAxisValue^[index+1]=TriangleAxisValue^[index])
do Inc(index);
    end;
    FindTriangleLevel:=index;
  end;
```

```
{-------------------------------------------------------

                BUILD Various File Routines

-------------------------------------------------------}


procedure Log(LogText: String);
begin
  if not LaserOff then begin
    {$I-}
    assign(HistoryFile,BuildHistoryFileName);
    {$I-}
    append(HistoryFile);
    if IOresult <> 0 then rewrite(HistoryFile);
    {$I+}
    if IOresult = 0 then begin
      writeln(HistoryFile,LogText);
      close(HistoryFile);
    end else beep;
  end;
end;

function EofVectorFile: Boolean;
begin
  EofVectorFile := eof(VectorFile) and (BufIndex >
VectorNumRead);
end;

procedure OpenFiles;
var
  FirstChar: Char;
label EasyLayer;
begin
  { look for either a .R or .L control file }
  if LayerCtrl then goto EasyLayer;
  assign(ControlFile,PartName+'.R');
```

```
  {$I-}
  reset(ControlFile);
  if IOresult = 0 then begin
    if eof(ControlFile) then begin
      SystemError('Error: Control File ' + PartName +
'.R empty.');
      halt(6);
    end;
  end else begin
EasyLayer:
    assign(ControlFile,PartName + '.L');
    reset(ControlFile);
    if IOresult<>0 then begin
      SystemError('No Control File for part ' +
PartName);
      halt(5);
    end else begin
      if eof(ControlFile) then begin
        SystemError('Error: Control File ' + PartName +
'.L empty.');
        halt(7);
      end else LayerCtrl := true;
    end;
  end;
    { look for the vector file }
  assign(VectorFile,PartName + '.V');
  reset(VectorFile,1);
  if IOresult<>0 then begin
    CantOpen(PartName + '.V');
    halt(8);
  end;
  if EofVectorFile then begin
    SystemError('Error: ' + PartName + '.V empty.');
    halt(9);
  end;
  {$I+}
  BlockRead(VectorFile,FirstChar,1,VectorNumRead);
```

```
  if FirstChar = #0 then VectorFileIsBinary := true
  else begin
    VectorFileIsBinary := false;
    close(VectorFile);
    reset(VectorFile,1);
  end;
end;

procedure ParseControlLineComments;
begin
  if (length(ControlLine)>0) and
(copy(ControlLine,1,6)='!MERGE') then begin
    i := pos('-STARTLAYER ',ControlLine);
    if (i>0) and (StartZ=0)
      then StartZ :=
LongVal(copy(ControlLine,i+12,5),dummy);
    i := pos('-ENDLAYER ',ControlLine);
    if (i>0) and (EndZ=65535)
      then EndZ :=
LongVal(copy(ControlLine,i+10,5),dummy);
    i := pos('-NUMLAYERS ',ControlLine);
    if (i>0) and (NumLayers=65535)
      then NumLayers :=
LongVal(copy(ControlLine,i+11,5),dummy);
  end;
end;

procedure GetLayerControlLine;
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      SameBlock := false;
      exit;
    end;
    readln(ControlFile,ControlLine);
    ParseControlLineComments;
```

```
  until (length(ControlLine)>0) and
(ControlLine[1]<>'!');
  ControlIndex := 1;
  UnitValInt(ControlLayer,ControlLine,ControlIndex);
  UnitStr(ControlType,ControlLine,ControlIndex);
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
{  writeln(SameBlock,' CLayer=',ControlLayer,'
CType=',ControlType,' CLine=',ControlLine); }
end;

procedure GetRangeControlLine;
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      exit;
    end;
    readln(ControlFile,ControlLine);
    ParseControlLineComments;
  until (length(ControlLine)>0) and
(ControlLine[1]<>'!');
end;

procedure ReadRange;
var
  Index,p: Integer;
  AtEnd: Boolean;
begin
  Index := 1;
  UnitValInt     (RangeZStart,   ControlLine,Index);
  UnitValInt     (RangeZEnd,     ControlLine,Index);
  UnitValShortInt(RangeZSpace,   ControlLine,Index);
  UnitValShortInt(RangeCureDepth,ControlLine,Index);
  AtEnd := false;
  BlockCount := 0;
  repeat
```

```
    GetRangeControlLine;
    if (ControlLine='') or (ControlLine[1] in
['0'..'9']) then AtEnd := true
    else begin
      inc(BlockCount);
      p := pos(',',ControlLine);
      if p=0 then p := length(ControlLine)+1;
      VectorTypes[BlockCount] :=
copy(ControlLine,1,p-1);
      BlockCmds [BlockCount] :=
NoQuotes(copy(ControlLine,p+1,255));
    end;
  until AtEnd;
end;

procedure ReSeekVectorFile;
begin
  if VectorFileLoc > FileSize(VectorFile)
    then VectorFileLoc := FileSize(VectorFile);  { force
eof condition }
  Seek(VectorFile,VectorFileLoc);
  BlockRead(VectorFile,VectorBuf,VectorRecordSize,
VectorNumRead);
  BufIndex := 1;
  BlockVectorCount := 0;
end;

function GetVectorCh: Char;
begin
  if not EofVectorFile then begin
    GetVectorCh := VectorBuf[BufIndex];
    inc(BufIndex);
    inc(VectorFileLoc);
    if BufIndex>VectorNumRead then begin
      BufIndex := 1;
      BlockRead(VectorFile,VectorBuf,VectorRecordSize,
VectorNumRead);
```

```
    end;
  end;
end;

function GetVectorByte: Byte;
begin
  if not EofVectorFile then GetVectorByte :=
ord(GetVectorCh)
  else GetVectorByte := 0;
end;

function GetVectorWord: Word;
var
  WordValue: Word;
begin
  WordValue := GetVectorByte;
  GetVectorWord := WordValue + GetVectorByte shl 8;
end;

function GetVectorLongInt: LongInt;
var
  LongIntValue: LongInt;
begin
  LongIntValue := GetVectorWord;
  GetVectorLongInt := LongIntValue + GetVectorWord shl
16;
end;

procedure GetVectorEntryAscii;
var
  Ch: Char;
  store: Integer;
begin
  repeat
    VectorLine := '';
    if EofVectorFile then begin
      SameBlock := false;
```

```
      exit;
    end;
    Ch := GetVectorCh;
    VectorLine[1] := Ch;
    store := 2;
    while (not EofVectorFile) and (Ch<>Cr) do begin
      Ch := GetVectorCh;
      VectorLine[store] := Ch;
      inc(store);
    end;
    VectorLine[0] := chr(store);
    { throw away line-feed }
    if not EofVectorFile then Ch := GetVectorCh;
    if VectorLine[1] in ['A'..'Z'] then begin
      BlockIndex := 1;
      UnitStr(VectorType,VectorLine,BlockIndex);
      UnitValInt(VectorLayer,VectorLine,BlockIndex);
    end;
  until (length(VectorLine)>0) and (VectorLine[1]<>'!');
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
{  writeln(SameBlock,' BLayer=',VectorLayer,'
BType=',VectorType,' BLine=',VectorLine); }
end;

procedure GetVectorEntryBinary;
begin
  if BlockVectorCount > 0 then begin
    Binary_X1 := GetVectorWord;
    Binary_Y1 := GetVectorWord;
    Binary_X2 := GetVectorWord;
    Binary_Y2 := GetVectorWord;
    dec(BlockVectorCount);
  end else begin
    VectorType := BlockMnemonics[GetVectorByte] +
chr(ord('0') + GetVectorByte);
    VectorLayer := GetVectorWord;
```

```
    BlockVectorCount := GetVectorLongInt;
  end;
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
end;

procedure GetVectorEntry;
begin
  if VectorFileIsBinary then GetVectorEntryBinary
    else GetVectorEntryAscii;
end;

{-------------------------------------------------------}

procedure BlockSkipAscii;
var
  Ch: Char;
  i,NumRead: Integer;
  Match: str10;
  SkipBuf: String;
begin
  Match := Lf+'L';
  while (VectorLayer < StartZ) and not EofVectorFile do
begin
    Seek(VectorFile,VectorFileLoc);
    BlockRead(VectorFile,SkipBuf[1],255,NumRead);
    i := pos(Match,SkipBuf);
    SkipBuf[0] := chr(255);
    while (i=0) and not(eof(VectorFile) or keypressed)
do begin
      inc(VectorFileLoc,NumRead);
      move(SkipBuf[236],SkipBuf[1],20);
      BlockRead(VectorFile,SkipBuf[20],235,NumRead);
      if keypressed then HandleKeyboardRequest;
      i := pos(Match,SkipBuf);
      if i>0 then dec(i,19);
    end;
```

```
    if i>0 then inc(VectorFileLoc,i)
    else VectorFileLoc := FileSize(VectorFile);
    ReSeekVectorFile;
    GetVectorEntry;
    if keypressed then HandleKeyboardRequest;
  end;
end;

procedure BlockSkipBinary;
type
  VectorBlockEntryType = record
    Block      : Byte;
    MergeSet   : Byte;
    Layer      : Word;
    VectorCount: LongInt;
  end;
var
  NumRead: Integer;
  VectorBlockEntry: VectorBlockEntryType;
begin
  Seek(VectorFile,VectorFileLoc);
  while (VectorLayer < StartZ) and not eof(VectorFile)
do begin
    BlockRead(VectorFile,VectorBlockEntry,SizeOf
(VectorBlockEntry),NumRead);
    if VectorBlockEntry.Layer < StartZ then begin
      inc(VectorFileLoc,8 + VectorBlockEntry.VectorCount
* 8);
      Seek(VectorFile,VectorFileLoc);
    end else VectorLayer := VectorBlockEntry.Layer;
    if keypressed then HandleKeyboardRequest;
  end;
  ReSeekVectorFile;
  GetVectorEntry;
end;

procedure BlockSkip;
```

```
begin
  if VectorFileIsBinary then BlockSkipBinary
    else BlockSkipAscii;
end;
```

```
{-------------------------------------------------------

                General Graphics Routines

-------------------------------------------------------}

procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn,0,ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x + len * TextWidth('A'),y + TextHeight(Text)
+ 1);
  SetColor(color);
  OutTextXY(x,y,Text);
end;

procedure VisualPage(Page: Integer);
begin
  SetVisualPage(Page);
  VisualPageNumber := Page;
end;

procedure BuildImage;
begin
```

```
  OnScreen := 1-OnScreen;
  SetActivePage(OnScreen);
end;

procedure ShowImage;
begin
  VisualPage(OnScreen);
end;

function GetInput(maxlen: Integer;Special:
SpecialKeyType;
  DefaultInput: String): String;
var
  temp: String;
  NoKey,First: Boolean;
  OrgColor,baseX,baseY,index: Integer;
begin
  OrgColor := GetColor;
  temp := '';
  index := 0;
  baseX := getX;
  baseY := getY;
  First := true;
  key := chr(0);
  VisualPage(OnScreen);
  SetColor(Cyan);
  OutText(DefaultInput);
  MoveTo(baseX,GetY);
  while key<>Cr do begin
    MoveTo(baseX+index*8,baseY);
    SetColor(Green);
    if First then OutText('_')
      else OverText('_',maxlen-index);
    First := false;
    NoKey := true;
    while NoKey do begin
      GetBuildKey;
```

```
    if Special = Menu then begin
      if not HandleSpecialKeys then NoKey := false;
    end else NoKey := false;
  end;
  MoveTo(baseX+index*8,baseY);
  SetColor(Black);
  OutText('_');
  First := false;
  MoveTo(baseX+index*8,baseY);
  SetColor(Cyan);
  if key in [Bs,Cr,Esc,' '..'~'] then begin
    if key=Bs then begin
      if index>0 then dec(index);
    end else if key=Esc then begin
      temp := DefaultInput;
      index := length(DefaultInput);
      key := Cr;
    end else if key in[' '..'~'] then begin
      OverText(key,1);
      if index<maxlen then inc(index);
      temp[index] := key;
    end;
  end;
  if ordkey = EndKey then begin
    MoveTo(baseX,baseY);
    OverText(DefaultInput,maxlen);
    temp := DefaultInput;
    index := length(DefaultInput);
  end;
end;
SetColor(Black);
MoveTo(baseX+index*8,baseY);
write('_');
SetColor(Cyan);
temp[0] := chr(index);
if temp = '' then GetInput := DefaultInput
  else GetInput := temp;
```

```
  SetColor(OrgColor);
end;

procedure GrAbort(Msg: string);
begin
  Writeln(Msg,': ',GraphErrorMsg(GraphResult));
  Halt(1);
end;

procedure InstallGraphics;
begin
  { Register all the drivers }
  if RegisterBGIdriver(@CGADriverProc) < 0 then
    GrAbort('CGA');
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    GrAbort('EGA/VGA');

  { Register all the fonts }
  if RegisterBGIfont(@SansSerifFontProc) < 0 then
    GrAbort('SansSerif');
  if RegisterBGIfont(@SmallFontProc) < 0 then
    GrAbort('Small');
  if RegisterBGIfont(@TriplexFontProc) < 0 then
    GrAbort('Triplex');

  GraphDriver := Detect;                { autodetect the
hardware }
  InitGraph(GraphDriver,GraphMode,'');  { activate
graphics }
  if GraphResult<>grOk then             { any errors? }
  begin
    Writeln('Graphics init error: ', GraphErrorMsg
(GraphDriver));
    Halt(1);
  end;
  VisualPage(1);
  ClearDevice;
```

```
  OnScreen := 1;  { start with Screen 0 }
end;


{-------------------------------------------------------

                BUILD Graphics Routines

-------------------------------------------------------}

{
  Print action information
}
procedure Action(ActionNum: BuildActionsType);
var
  i: Integer;
  Text : String;
begin
  { Stop all counters except Total (0) }
  for i:=1 to MAX_BENCH_ITEMS do BenchStop(i);
  Case ActionNum of
    ACTION_LOADING    : begin
                        Text := 'LOADING';
                        BenchStart(ord(BENCH_BUILD_
LOADING));
                        end;
    ACTION_DRAWING    : begin
                        Text := 'DRAWING';
                        BenchStart(ord(BENCH_BUILD_
DRAWING));
                        end;
    ACTION_SURFACING : begin
                        Text := 'SURFACING';
                        BenchStart(ord(BENCH_BUILD_
DRAWING));
                        end;
    ACTION_DIPPING    : begin
```

```
                        Text := 'DIPPING';
                        BenchStart(ord(BENCH_BUILD_
DIPPING));
                        end;
    ACTION_LEVELING  : begin
                        Text := 'LEVELING';
                        BenchStart(ord(BENCH_BUILD_
LEVELING));
                        end;
    ACTION_PAUSED    : begin
                        Text := 'PAUSED';
                        BenchStart(ord(BENCH_BUILD_
PAUSED));
                        end;
    ACTION_SKIPPING  : begin
                        Text := 'SKIPPING';
                        BenchStart(ord(BENCH_BUILD_
SKIPPING));
                        end;
    ACTION_ANALYZING : begin
                        Text := 'ANALYZING';
                        BenchStart(ord(BENCH_BUILD_
ANALYZE));
                        end;
    ACTION_PARTCOMPL : begin
                        Text := 'PART COMPLETED';
                        BenchStop(ord(BENCH_BUILD_
TOTAL));
                        end;
    ACTION_SWEEPING  : begin
                        Text := 'SWEEPING';
                        BenchStart(ord(BENCH_BUILD_
SWEEPING));
                        end;
    else                begin
                        Text := 'UNKNOWN';
                        end;
```

```
      end;
    if OpMode=TextOp then write(Text,' ')
    else begin
      MoveTo(40*8,7*14);
      SetColor(Green);
      OverText(Text,20);
      SetColor(Cyan);
    end;
  end;
```

```
{-------------------------------------------------------

                BUILD Keyboard Routines

-------------------------------------------------------}

function STEREO(LaserCommand: String): Integer; forward;
procedure ShowError(Error: Integer);            forward;

procedure CleanUp;
begin
  if STEREO('SP 100;EC;EC')<>0 then begin end;   { don't
show error }
  if (not LaserOff) and (not SetForLaserOff) then
    Exec('C:\COMMAND.COM','/C SERVICE LASER OFF
+60/i/t');
  if OpMode=GraphicsOp then TextMode(C80);
  while keypressed do key := ReadKey;
end;

procedure Abort;
begin
  CleanUp;
  halt(127);
end;

procedure Quit;
begin
  CleanUp;
  halt(0);
end;

procedure GetBuildKey;
begin
  Escape := false;
  key := ReadKey;
  if key = Ctrl C then Abort;
```

```
  if (key = #0) and keypressed then begin
    key2 := ReadKey;
    ordkey := 1000 + ord(Key2);
  end else begin
    key2 := #0;
    ordkey := ord(key);
  end;
  if key = Esc then Escape := true;
end;

procedure VisualPage(Page: Integer); forward;
procedure OptionsMenu; forward;

function HandleSpecialKeys: Boolean;
begin
  HandleSpecialKeys := false;   { true mean special key
pressed }
  if OpMode = GraphicsOp then begin
    if ordkey = F1 then begin
      VisualPage(0);
      HandleSpecialKeys := true;
    end else if ordkey = F2 then begin
      VisualPage(1);
      HandleSpecialKeys := true;
    end else if ordkey = Ctrl F6 then begin
      port[$305] := port[$305] or  1;   { reset NMI on
failed AMD 80286 }
      port[$305] := port[$305] and $FF-1;
      port[$61]  := port[$61]  or  8;
      port[$61]  := port[$61]  and $FF-8;
    end else if ordkey = Ctrl F10 then begin
      OptionsMenu;
      HandleSpecialKeys := true;
    end;
  end;
end;
```

```
procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    GetBuildKey;
    if key = CtrlS then GetBuildKey;   { Pause }
    if HandleSpecialKeys then begin end;
  end;
end;
```

```
{-------------------------------------------------------

              BUILD Laser Controller Routines

-------------------------------------------------------}

{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
```

```
    if LaserOk then LaserPresent := true
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{
  Check for Memory-Resident STEREO Stereolithography
Device Driver
}
function StereoOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;
```

```
function StereoPresent: Boolean;
  begin
    if StereoOk then StereoPresent := true
    else begin
      StereoPresent := false;
      SystemError('STEREO driver not installed.');
    end;
  end;

procedure ShowError(Error: Integer);
begin
  case Error of
      0: ;   { no error }
      1: PrintError('1. Laser interface card not
found');
      2: PrintError('2. Laser channel not open');
      3: PrintError('3. Background Command Queue full
timeout');
      4: PrintError('4. Invalid interrupt time');
      5: PrintError('5. Not in block mode');
      6: PrintError('6. Exec Queue limit timeout');
      7: PrintError('7. Invalid timer channel
selected');
    127: PrintError('127. Bad SIH command parameter');
    128: PrintError('128. Unrecognized STEREO command');
    129: PrintError('129. Invalid STEREO communications
request');
    255: PrintError('255. Internal error (please
report)');
  else PrintError(IntStr(Error)+'. Unrecognized SIH
error');
  end;
end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
          Pass Command on to Memory Resident STEREO
Device Driver

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

function STEREO(LaserCommand: String): Integer;
begin
  STEREO := 0;
  if length(LaserCommand)>0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      writeln('   Laser>',LaserCommand);
    end;
    if not LaserOff then begin
      if LaserCommand[1]<>'!' then with regs do begin
        LaserCommand := LaserCommand + chr(0);
        AX := $00;   { ASCII text format }
        BX := Ofs(LaserCommand)+1;   { pass pointer to
string }
        ES := Seg(LaserCommand);
        Intr(STEREOVECTOR,regs);
        STEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if STEREO_KeyCheck and keypressed then
HandleKeyboardRequest;
end;

function BinarySTEREO(CommandBuffer: Pointer): Integer;
var
  BufSeg,BufOfs: Word;
begin
  BinarySTEREO := 0;
  BufSeg := seg(CommandBuffer^);
  BufOfs := ofs(CommandBuffer^);
```

```
  if Mem[BufSeg:BufOfs] > 0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      write('   BinaryLaser>');
      for i:=0 to 11 do
write(HexByte(Mem[BufSeg:BufOfs+i]),' ');
      writeln;
    end;
    if not LaserOff then begin
      with regs do begin
        AX := $01;   { binary format }
        BX := BufOfs; { pass pointer to binary command }
        ES := BufSeg;
        Intr(STEREOVECTOR,regs);
        BinarySTEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if STEREO_KeyCheck and keypressed then
HandleKeyboardRequest;
end;

{-------------------------------------------------------

  Send execute command to the Stereo driver
}
procedure LaserExecute;
const
  BinaryCommand: array[1..5] of Byte =
    (STEREO_EX,STEREO_CL,STEREO_EC,STEREO_EC,0);
begin
  ShowError(BinarySTEREO(@BinaryCommand));
  LaserQueueBlockSize := 0;
end;

{-------------------------------------------------------
```

```
  Send laser control command(s) to the Stereo driver
}
procedure SendLaserCommand(cmd: String);
begin
  ShowError(STEREO(cmd));
  LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
LaserExecute;
end;

procedure SendBinaryLaserCommand(BufPtr: Pointer);
begin
  ShowError(BinarySTEREO(BufPtr));
  LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
LaserExecute;
end;

{-------------------------------------------------------

  Pick Out and Execute BUILD-specific Command
}
procedure PickOutBuildCommand(var cmd: String);
var
  op: str4;
  k,index: Integer;
  arg: FloatType;
begin
  repeat
    op := upper(copy(cmd,1,2));
    index := 1;
    while (index<=BuildCommandsCount) and
(pos(BuildCommands[index],op)=0)
      do inc(index);
    if index<=BuildCommandsCount then begin
      k := 3;
```

```
      while (k<length(cmd)) and (cmd[k]<>';') do inc(k);
      inc(k);
      arg := RealVal(copy(cmd,3,k-3),code);
      case index of
        1: {ZWait}  RelaxationTime := LibTrunc(arg);
        2: {ZDepth} DipDepth       := arg;
        3: {ZVeloc} ZTableVeloc    := arg;
        4: {ZAccel} ZTableAccel    := arg;
      end;
      cmd := copy(cmd,k,199);
    end;
  until index > BuildCommandsCount;
end;

{-------------------------------------------------------

  Process both BUILD and STEREO commands

  BUILD and STEREO commands are broken apart and
executed in the proper sequence, but by the two
different programs.
}
procedure ProcessCommands(cmd: String);
var
  OrgCmd: String;
  i,j,BuildIndex: Integer;
begin
  cmd := upper(cmd);
  repeat
    OrgCmd := cmd;
    i := length(OrgCmd) + 1;
    for j:=1 to BuildCommandsCount do begin
      BuildIndex := pos(BuildCommands[j],OrgCmd);
      if (BuildIndex > 0) and (BuildIndex < i) then i :=
BuildIndex;
    end;
    cmd := copy(OrgCmd,1,i-1);
```

```
    if length(cmd)>0 then SendLaserCommand(cmd);
    cmd := copy(OrgCmd,i,255);
    if length(cmd)>0 then PickOutBuildCommand(cmd);
  until length(cmd)=0;
end;

{-------------------------------------------------------

{
  Find Block Type in ParVectorType array.  If there,
send the corresponding ParStereoDef default string to
Stereo.
}
procedure SetBlockDefaults;
begin
  i := 1;
  while (ParVectorType[i]<>ControlType) and
(i<=ParCount) do inc(i);
  if i<=ParCount then
    ProcessCommands(ParStereoDef[i]);
end;

{-------------------------------------------------------

function HardwareLaserOn: Boolean;
const
  PowerPort = $304;
  _LASER_IN = $80;
begin
  if port[PowerPort + 2] and _LASER_IN = 0 then
HardwareLaserOn := true
    else HardwareLaserOn := false;
end;
```

```
procedure BenchInit;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    for i:=1 to BENCH_ITEMS do begin
      Bench_Item_Total[i]:=0;
      Bench_Item_Stamp[i]:=0;
    end;
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    writeln(BenchFile,'Benchmark Started for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
     lose(BenchFile);
      nch_Start_DOW := DOW;
  end;

function GetS100Time : LongInt;
  var
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    if DOW<Bench_Start_DOW then Inc(DOW,7);
    GetS100Time := S100 + LongInt(Sec)*100 +
LongInt(Min)*6000 + Hr*360000 +
                  (DOW-Bench_Start_DOW)*8640000;
  end;


procedure BenchStart(Item : Word);
```

```
  begin
    if Bench_Item_Stamp[Item]=0 then
Bench_Item_Stamp[Item]:=GetS100Time;
  end;

procedure BenchStop(Item : Word);
  begin
    if Bench_Item_Stamp[Item]>0 then begin
      Inc(Bench_Item_Total[Item],GetS100Time -
Bench_Item_Stamp[Item]);
      Bench_Item_Stamp[Item]:=0;
    end;
  end;

procedure BenchEnd;
  var
    BenchFile : Text;
    Hr,Min,Sec,S100,Mon,Day,Yr,DOW : Word;
    i : Integer;
  begin
    GetDate(Yr,Mon,Day,DOW);
    GetTime(Hr,Min,Sec,S100);
    assign(BenchFile,BENCH_FILE);
    {$I-} Append(BenchFile); {$I+}
    if IOResult<>0 then rewrite(BenchFile);
    for i:=1 to BENCH_ITEMS do
      if Bench_Item_Stamp[i]>0 then begin
        BenchStop(i);
        writeln(BenchFile,' Total ',Bench_Item_Title[i],

                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec RUNNING');
      end else
        writeln(BenchFile,' Total ',Bench_Item_Title[i],

                          ' Time =
',Bench_Item_Total[i]/100:1:2,' sec');
```

```
    writeln(BenchFile,'Benchmark Ended   for BUILD.EXE
on ',
            Mon,'/',Day,'/',Yr,' at ',
            Hr,':',Min,':',Sec,'.',S100);
    Close(BenchFile);
  end;
```

```
{-------------------------------------------------------

                BUILD Various File Routines

-------------------------------------------------------}

procedure Log(LogText: String);
begin
  if not LaserOff then begin
    {$I-}
    assign(HistoryFile,BuildHistoryFileName);
    {$I-}
    append(HistoryFile);
    if IOresult <> 0 then rewrite(HistoryFile);
    {$I+}
    if IOresult = 0 then begin
      writeln(HistoryFile,LogText);
      close(HistoryFile);
    end else beep;
  end;
end;

function EofVectorFile: Boolean;
begin
  EofVectorFile := eof(VectorFile) and (BufIndex >
VectorNumRead);
end;

procedure OpenFiles;
var
  FirstChar: Char;
label EasyLayer;
begin
  { look for either a .R or .L control file }
  if LayerCtrl then goto EasyLayer;
  assign(ControlFile,PartName+'.R');
  {$I-}
```

```
  reset(ControlFile);
  if IOresult = 0 then begin
    if eof(ControlFile) then begin
      SystemError('Error: Control File ' + PartName +
'.R empty.');
      halt(6);
    end;
  end else begin
EasyLayer:
    assign(ControlFile,PartName + '.L');
    reset(ControlFile);
    if IOresult<>0 then begin
      SystemError('No Control File for part ' +
PartName);
      halt(5);
    end else begin
      if eof(ControlFile) then begin
        SystemError('Error: Control File ' + PartName +
'.L empty.');
        halt(7);
      end else LayerCtrl := true;
    end;
  end;
    { look for the vector file }
  assign(VectorFile,PartName + '.V');
  reset(VectorFile,1);
  if IOresult<>0 then begin
    CantOpen(PartName + '.V');
    halt(8);
  end.
  if  ofVectorFile then begin
    SystemError('Error: ' + PartName + '.V empty.');
    halt(9);
  end;
  {$I+}
  BlockRead(VectorFile,FirstChar,1,VectorNumRead);
  if FirstChar = #0 then VectorFileIsBinary := true
```

```
  else begin
    VectorFileIsBinary := false;
    close(VectorFile);
    reset(VectorFile,1);
  end;
end;

procedure ParseControlLineComments;
begin
  if (length(ControlLine)>0) and
(copy(ControlLine,1,6)='!MERGE') then begin
    i := pos('-STARTLAYER ',ControlLine);
    if (i>0) and (StartZ=0)
      then StartZ :=
LongVal(copy(ControlLine,i+12,5),dummy);
    i := pos('-ENDLAYER ',ControlLine);
    if (i>0) and (EndZ=65535)
      then EndZ :=
LongVal(copy(ControlLine,i+10,5),dummy);
    i := pos('-NUMLAYERS ',ControlLine);
    if (i>0) and (NumLayers=65535)
      then NumLayers :=
LongVal(copy(ControlLine,i+11,5),dummy);
  end;
end;

procedure GetLayerControlLine;
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      SameBlock := false;
      exit;
    end;
    readln(ControlFile,ControlLine);
    ParseControlLineComments;
```

```
  until (length(ControlLine)>0) and
(ControlLine[1]<>'!');
  ControlIndex := 1;
  UnitValInt(ControlLayer,ControlLine,ControlIndex);
  UnitStr(ControlType,ControlLine,ControlIndex);
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
{  writeln(SameBlock,' CLayer=',ControlLayer,'
CType=',ControlType,' CLine=',ControlLine); }
end;

procedure GetRangeControlLine;
begin
  repeat
    if eof(ControlFile) then begin
      ControlLine := '';
      exit;
    end;
    readln(ControlFile,ControlLine);
    ParseControlLineComments;
  until (length(ControlLine)>0) and
(ControlLine[1]<>'!');
end;

procedure ReadRange;
var
  Index,p: Integer;
  AtEnd: Boolean;
begin
  Index := 1;
  UnitValInt     (RangeZStart,    ControlLine,Index);
  UnitValInt     (RangeZEnd,      ControlLine,Index);
  UnitValShortInt(RangeZSpace,    ControlLine,Index);
  UnitValShortInt(RangeCureDepth,ControlLine,Index);
  AtEnd := false;
  BlockCount := 0;
  repeat
```

```
    GetRangeControlLine;
    if (ControlLine='') or (ControlLine[1] in
['0'..'9']) then AtEnd := true
    else begin
      inc(BlockCount);
      p := pos(',',ControlLine);
      if p=0 then p := length(ControlLine)+1;
      VectorTypes[BlockCount] :=
copy(ControlLine,1,p-1);
      BlockCmds [BlockCount] :=
NoQuotes(copy(ControlLine,p+1,255));
    end;
  until AtEnd;
end;

procedure ReSeekVectorFile;
begin
  if VectorFileLoc > FileSize(VectorFile)
    then VectorFileLoc := FileSize(VectorFile);  { force
eof condition }
  Seek(VectorFile,VectorFileLoc);
  BlockRead(VectorFile,VectorBuf,VectorRecordSize,
VectorNumRead);
  BufIndex := 1;
  BlockVectorCount := 0;
end;

function GetVectorCh: Char;
begin
  if not EofVectorFile then begin
    GetVectorCh := VectorBuf[BufIndex];
    inc(BufIndex);
    inc(VectorFileLoc);
    if BufIndex>VectorNumRead then begin
      BufIndex := 1;
      BlockRead(VectorFile,VectorBuf,VectorRecordSize,
VectorNumRead);
```

```
    end;
  end;
end;

function GetVectorByte: Byte;
begin
  if not EofVectorFile then GetVectorByte :=
ord(GetVectorCh)
  else GetVectorByte := 0;
end;

function GetVectorWord: Word;
var
  WordValue: Word;
begin
  WordValue := GetVectorByte;
  GetVectorWord := WordValue + GetVectorByte shl 8;
end;

function GetVectorLongInt: LongInt;
var
  LongIntValue: LongInt;
begin
  LongIntValue := GetVectorWord;
  GetVectorLongInt := LongIntValue + GetVectorWord shl
16;
end;

procedure GetVectorEntryAscii;
var
  Ch: Char;
  store: Integer;
begin
  repeat
    VectorLine := '';
    if EofVectorFile then begin
      SameBlock := false;
```

```
      exit;
    end;
    Ch := GetVectorCh;
    VectorLine[1] := Ch;
    store := 2;
    while (not EofVectorFile) and (Ch<>Cr) do begin
      Ch := GetVectorCh;
      VectorLine[store] := Ch;
      inc(store);
    end;
    VectorLine[0] := chr(store);
    { throw away line-feed }
    if not EofVectorFile then Ch := GetVectorCh;
    if VectorLine[1] in ['A'..'Z'] then begin
      BlockIndex := 1;
      UnitStr(VectorType,VectorLine,BlockIndex);
      UnitValInt(VectorLayer,VectorLine,BlockIndex);
    end;
  until (length(VectorLine)>0) and (VectorLine[1]<>'!');
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
{  writeln(SameBlock,' BLayer=',VectorLayer,'
BType=',VectorType,' BLine=',VectorLine); }
end;

procedure GetVectorEntryBinary;
begin
  if BlockVectorCount > 0 then begin
    Binary_X1 := GetVectorWord;
    Binary_Y1 := GetVectorWord;
    Binary_X2 := GetVectorWord;
    Binary_Y2 := GetVectorWord;
    dec(BlockVectorCount);
  end else begin
    VectorType := BlockMnemonics[GetVectorByte] +
chr(ord('0') + GetVectorByte);
    VectorLayer := GetVectorWord;
```

```
    BlockVectorCount := GetVectorLongInt;
  end;
  SameBlock := (VectorLayer = ControlLayer) and
(VectorType = ControlType);
end;

procedure GetVectorEntry;
begin
  if VectorFileIsBinary then GetVectorEntryBinary
    else GetVectorEntryAscii;
end;

{-------------------------------------------------------

procedure BlockSkipAscii;
var
  Ch: Char;
  i,NumRead: Integer;
  Match: str10;
  SkipBuf: String;
begin
  Match := Lf+'L';
  while (VectorLayer < StartZ) and not EofVectorFile do
begin
    Seek(VectorFile,VectorFileLoc);
    BlockRead(VectorFile,SkipBuf[1],255,NumRead);
    i := pos(Match,SkipBuf);
    SkipBuf[0] := chr(255);
    while (i=0) and not(eof(VectorFile) or keypressed)
do begin
      inc(Ve  orFileLoc,NumRead);
      move(S  pBuf[236],SkipBuf[1],20);
      BlockR..ad(VectorFile,SkipBuf[20],235,NumRead);
      if keypressed then HandleKeyboardRequest;
      i := pos(Match,SkipBuf);
      if i>0 then dec(i,19);
    end;
```

```
    if i>0 then inc(VectorFileLoc,i)
    else VectorFileLoc := FileSize(VectorFile);
    ReSeekVectorFile;
    GetVectorEntry;
    if keypressed then HandleKeyboardRequest;
  end;
end;

procedure BlockSkipBinary;
type
  VectorBlockEntryType = record
    Block      : Byte;
    MergeSet   : Byte;
    Layer      : Word;
    VectorCount: LongInt;
  end;
var
  NumRead: Integer;
  VectorBlockEntry: VectorBlockEntryType;
begin
  Seek(VectorFile,VectorFileLoc);
  while (VectorLayer < StartZ) and not eof(VectorFile)
do begin
    BlockRead(VectorFile,VectorBlockEntry,SizeOf
(VectorBlockEntry),NumRead);
    if VectorBlockEntry.Layer < StartZ then begin
      inc(VectorFileLoc,8 + VectorBlockEntry.VectorCount
* 8);
      Seek(VectorFile,VectorFileLoc);
    end else VectorLayer := VectorBlockEntry.Layer;
    if keypressed then HandleKeyboardRequest;
  end;
  ReSeekVectorFile;
  GetVectorEntry;
end;

procedure BlockSkip;
```

```
begin
  if VectorFileIsBinary then BlockSkipBinary
    else BlockSkipAscii;
end;
```

```
{-------------------------------------------------------

                General Graphics Routines

-------------------------------------------------------}

procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn,0,ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x + len * TextWidth('A'),y + TextHeight(Text)
+ 1);
  SetColor(color);
  OutTextXY(x,y,Text);
end;

procedure VisualPage(Page: Integer);
begin
  SetVisualPage(Page);
  VisualPageNumber := Page;
end;

procedure BuildImage;
begin
```

```
  OnScreen := 1-OnScreen;
  SetActivePage(OnScreen);
end;

procedure ShowImage;
begin
  VisualPage(OnScreen);
end;

function GetInput(maxlen: Integer;Special:
SpecialKeyType;
  DefaultInput: String): String;
var
  temp: String;
  NoKey,First: Boolean;
  OrgColor,baseX,baseY,index: Integer;
begin
  OrgColor := GetColor;
  temp := '';
  index := 0;
  baseX := getX;
  baseY := getY;
  First := true;
  key := chr(0);
  VisualPage(OnScreen);
  SetColor(Cyan);
  OutText(DefaultInput);
  MoveTo(baseX,GetY);
  while key<>Cr do begin
    MoveTo(baseX+index*8,baseY);
    SetColor(Green);
    if First then OutText('_')
      else OverText('_',maxlen-index);
    First := false;
    NoKey := true;
    while NoKey do begin
      GetBuildKey;
```

```
      if Special = Menu then begin
        if not HandleSpecialKeys then NoKey := false;
      end else NoKey := false;
    end;
    MoveTo(baseX+index*8,baseY);
    SetColor(Black);
    OutText('_');
    First := false;
    MoveTo(baseX+index*8,baseY);
    SetColor(Cyan);
    if key in [Bs,Cr,Esc,' '..'~'] then begin
      if key=Bs then begin
        if index>0 then dec(index);
      end else if key=Esc then begin
        temp := DefaultInput;
        index := length(DefaultInput);
        key := Cr;
      end else if key in[' '..'~'] then begin
        OverText(key,1);
        if index<maxlen then inc(index);
        temp[index] := key;
      end;
    end;
    if ordkey = EndKey then begin
      MoveTo(baseX,baseY);
      OverText(DefaultInput,maxlen);
      temp := DefaultInput;
      index := length(DefaultInput);
    end;
  end;
  SetColor(Black);
  MoveTo(baseX+index*8,baseY);
  write('_');
  SetColor(Cyan);
  temp[0] := chr(index);
  if temp = '' then GetInput := DefaultInput
    else GetInput := temp;
```

```
  SetColor(OrgColor);
end;

procedure GrAbort(Msg: string);
begin
  Writeln(Msg,': ',GraphErrorMsg(GraphResult));
  Halt(1);
end;

procedure InstallGraphics;
begin
  { Register all the drivers }
  if RegisterBGIdriver(@CGADriverProc) < 0 then
    GrAbort('CGA');
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    GrAbort('EGA/VGA');

  { Register all the fonts }
  if RegisterBGIfont(@SansSerifFontProc) < 0 then
    GrAbort('SansSerif');
  if RegisterBGIfont(@SmallFontProc) < 0 then
    GrAbort('Small');
  if RegisterBGIfont(@TriplexFontProc) < 0 then
    GrAbort('Triplex');

  GraphDriver := Detect;                    { autodetect
the hardware }
  InitGraph(GraphDriver,GraphMode,'');      { activate
graphics }
  if GraphResult<>grOk then                 { any errors?
}
  begin
    Writeln('Graphics init error: ',
GraphErrorMsg(GraphDriver));
    Halt(1);
  end;
  VisualPage(1);
```

```
  ClearDevice;
  OnScreen := 1;  { start with Screen 0 }
end;


{-------------------------------------------------------

                BUILD Graphics Routines

-------------------------------------------------------}

{
  Print action information
}
procedure Action(ActionNum: BuildActionsType);
var
  i: Integer;
  Text : String;
begin
  { Stop all counters except Total (0) }
  for i:=1 to MAX_BENCH_ITEMS do BenchStop(i);
  Case ActionNum of
    ACTION_LOADING   : begin
                         Text := 'LOADING';

BenchStart(ord(BENCH_BUILD_LOADING));
                         end;
    ACTION_DRAWING   : begin
                         Text := 'DRAWING';
                         BenchStart(ord
(BENCH_BUILD_DRAWING));
                         end;
    ACTION_SURFACING : begin
                         Text := 'SURFACING';
                         BenchStart(ord
(BENCH_BUILD_DRAWING));
                         end;
```

```
    ACTION_DIPPING    : begin
                        Text := 'DIPPING';
                        BenchStart(ord
(BENCH_BUILD_DIPPING));
                        end;
    ACTION_LEVELING   : begin
                        Text := 'LEVELING';
                        BenchStart(ord
(BENCH_BUILD_LEVELING));
                        end;
    ACTION_PAUSED     : begin
                        Text := 'PAUSED';
                        BenchStart(ord
(BENCH_BUILD_PAUSED));
                        end;
    ACTION_SKIPPING   : begin
                        Text := 'SKIPPING';
                        BenchStart(ord
(BENCH_BUILD_SKIPPING));
                        end;
    ACTION_ANALYZING : begin
                        Text := 'ANALYZING';
                        BenchStart(ord
(BENCH_BUILD_ANALYZE));
                        end;
    ACTION_PARTCOMPL : begin
                        Text := 'PART COMPLETED';
                        BenchStop(ord
(BENCH_BUILD_TOTAL));
                        end;
    ACTION_SWEEPING   : begin
                        Text := 'SWEEPING';
                        BenchStart(ord
(BENCH_BUILD_SWEEPING));
                        end;
    else                begin
                        Text := 'UNKNOWN';
```

```
                        end;
      end;
    if OpMode=TextOp then write(Text,' ')
    else begin
      MoveTo(40*8,7*14);
      SetColor(Green);
      OverText(Text,20);
      SetColor(Cyan);
    end;
  end;
```

```
{-------------------------------------------------------

                  BUILD Keyboard Routines

-------------------------------------------------------}

function STEREO(LaserCommand: String): Integer; forward;
procedure ShowError(Error: Integer);            forward;

procedure CleanUp;
begin
  if STEREO('SP 100;EC;EC')<>0 then begin end;   { don't
show error }
  if (not LaserOff) and (not SetForLaserOff) then
    Exec('C:\COMMAND.COM','/C SERVICE LASER OFF
+60/i/t');
  if OpMode=GraphicsOp then TextMode(C80);
  while keypressed do key := ReadKey;
end;

procedure Abort;
begin
  CleanUp;
  halt(127);
end;

procedure Quit;
begin
  CleanUp;
  halt(0);
end;

procedure GetBuildKey;
begin
  Escape := false;
  key := ReadKey;
  if key = Ctrl C then Abort;
```

```
  if (key = #0) and keypressed then begin
    key2 := ReadKey;
    ordkey := 1000 + ord(Key2);
  end else begin
    key2 := #0;
    ordkey := ord(key);
  end;
  if key = Esc then Escape := true;
end;

procedure VisualPage(Page: Integer); forward;
procedure OptionsMenu; forward;

function HandleSpecialKeys: Boolean;
begin
  HandleSpecialKeys := false;   { true mean special key
pressed }
  if OpMode = GraphicsOp then begin
    if ordkey = F1 then begin
      VisualPage(0);
      HandleSpecialKeys := true;
    end else if ordkey = F2 then begin
      VisualPage(1);
      HandleSpecialKeys := true;
    end else if ordkey = CtrlF6 then begin
      port[$305] := port[$305] or  1;   { reset NMI on
failed AMD 80286 }
      port[$305] := port[$305] and $FF-1;
      port[$61]  := port[$61]  or  8;
      port[$61]  := port[$61]  and $FF-8;
    end else if ordkey = CtrlF10 then begin
      OptionsMenu;
      HandleSpecialKeys := true;
    end;
  end;
end;
```

```
procedure HandleKeyboardRequest;
begin
  while keypressed do begin
    GetBuildKey;
    if key = CtrlS then GetBuildKey;   { Pause }
    if HandleSpecialKeys then begin end;
  end;
end;
```

```
{-------------------------------------------------------

              BUILD Laser Controller Routines

-------------------------------------------------------}

{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
```

```
    if LaserOk then LaserPresent := true
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{
  Check for Memory-Resident STEREO Stereolithography
Device Driver
}
function StereoOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;
```

```
function StereoPresent: Boolean;
  begin
    if StereoOk then StereoPresent := true
    else begin
      StereoPresent := false;
      SystemError('STEREO driver not installed.');
    end;
  end;

procedure ShowError(Error: Integer);
begin
  case Error of
      0: ;   { no error }
      1: PrintError('1. Laser interface card not
found');
      2: PrintError('2. Laser channel not open');
      3: PrintError('3. Background Command Queue full
timeout');
      4: PrintError('4. Invalid interrupt time');
      5: PrintError('5. Not in block mode');
      6: PrintError('6. Exec Queue limit timeout');
      7: PrintError('7. Invalid timer channel
selected');
    127: PrintError('127. Bad SIH command parameter');
    128: PrintError('128. Unrecognized STEREO command');

    129: PrintError('129. Invalid STEREO communications
request');
    255: PrintError('255. Internal error (please
report)');
  else PrintError(IntStr(Error)+'. Unrecognized SIH
error');
  end;
end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
          Pass Command on to Memory Resident STEREO
Device Driver

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

function STEREO(LaserCommand: String): Integer;
begin
  STEREO := 0;
  if length(LaserCommand)>0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      writeln('   Laser>',LaserCommand);
    end;
    if not LaserOff then begin
      if LaserCommand[1]<>'!' then with regs do begin
        LaserCommand := LaserCommand + chr(0);
        AX := $00;   { ASCII text format }
        BX := Ofs(LaserCommand)+1;   { pass pointer to
string }
        ES := Seg(LaserCommand);
        Intr(STEREOVECTOR,regs);
        STEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if STEREO_KeyCheck and keypressed then
HandleKeyboardRequest;
end;

function BinarySTEREO(CommandBuffer: Pointer): Integer;
var
  BufSeg,BufOfs: Word;
begin
  BinarySTEREO := 0;
  BufSeg := seg(CommandBuffer^);
  BufOfs := ofs(CommandBuffer^);
```

```
  if Mem[BufSeg:BufOfs] > 0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      write('   BinaryLaser>');
      for i:=0 to 11 do write(HexByte(Mem[BufSeg:
BufOfs+i]),' ');
      writeln;
    end;
    if not LaserOff then begin
      with regs do begin
        AX := $01;   { binary format }
        BX := BufOfs; { pass pointer to binary command }
        ES := BufSeg;
        Intr(STEREOVECTOR,regs);
        BinarySTEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if STEREO_KeyCheck and keypressed then
HandleKeyboardRequest;
end;

{-------------------------------------------------------

  Send execute command to the Stereo driver
}
procedure LaserExecute;
const
  BinaryCommand: array[1..5] of Byte =
    (STEREO_EX,STEREO_CL,STEREO_EC,STEREO_EC,0);
begin
  ShowError(BinarySTEREO(@BinaryCommand));
  LaserQueueBlockSize := 0;
end;

-------------------------------------------------------}
```

```
  Send laser control command(s) to the Stereo driver
}
procedure SendLaserCommand(cmd: String);
begin
  ShowError(STEREO(cmd));
  LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
LaserExecute;
end;

procedure SendBinaryLaserCommand(BufPtr: Pointer);
begin
  ShowError(BinarySTEREO(BufPtr));
  LaserQueueBlockSize := LaserQueueBlockSize + QSize;
  if LaserQueueBlockSize > MaxLaserQueueBlockSize then
LaserExecute;
end;

{-------------------------------------------------------

  Pick Out and Execute BUILD-specific Command
}
procedure PickOutBuildCommand(var cmd: String);
var
  op: str4;
  k,index: Integer;
  arg: FloatType;
begin
  repeat
    op := upper(copy(cmd,1,2));
    index := 1;
    while (index<=BuildCommandsCount) and
(pos(BuildCommands[index],op)=0)
      do inc(index);
    if index<=BuildCommandsCount then begin
      k := 3;
```

```
      while (k<length(cmd)) and (cmd[k]<>';') do inc(k);
      inc(k);
      arg := RealVal(copy(cmd,3,k-3),code);
      case index of
        1: {ZWait}  RelaxationTime := LibTrunc(arg);
        2: {ZDepth} DipDepth       := arg;
        3: {ZVeloc} ZTableVeloc    := arg;
        4: {ZAccel} ZTableAccel    := arg;
      end;
      cmd := copy(cmd,k,199);
    end;
  until index > BuildCommandsCount;
end;

-------------------------------------------------------}

  Process both BUILD and STEREO commands

  BUILD and STEREO commands are broken apart and
executed in the proper sequence, but by the two
different programs.
}
procedure ProcessCommands(cmd: String);
var
  OrgCmd: String;
  i,j,BuildIndex: Integer;
begin
  cmd := upper(cmd);
  repeat
    OrgCmd := cmd;
    i := length(OrgCmd) + 1;
    for j:=1 to BuildCommandsCount do begin
      BuildIndex := pos(BuildCommands[j],OrgCmd);
      if (BuildIndex > 0) and (BuildIndex < i) then i :=
BuildIndex;
    end;
    cmd := copy(OrgCmd,1,i-1);
```

```
    if length(cmd)>0 then SendLaserCommand(cmd);
    cmd := copy(OrgCmd,i,255);
    if length(cmd)>0 then PickOutBuildCommand(cmd);
  until length(cmd)=0;
end;

{-------------------------------------------------------

{
  Find Block Type in ParVectorType array.  If there,
send the corresponding ParStereoDef default string to
Stereo.
}
procedure SetBlockDefaults;
begin
  i := 1;
  while (ParVectorType[i]<>ControlType) and
(i<=ParCount) do inc(i);
  if i<=ParCount then
    ProcessCommands(ParStereoDef[i]);
end;

{-------------------------------------------------------}

function HardwareLaserOn: Boolean;
const
  PowerPort = $304;
  _LASER_IN = $80;
begin
  if port[PowerPort + 2] and _LASER_IN = 0 then
HardwareLaserOn := true
    else HardwareLaserOn := false;
end;
```

```
{-------------------------------------------------------

                BUILD Time/Date Routines

-------------------------------------------------------}

{
  Pad time element with left space if necessary
}
function PadSp(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then PadSp := ' ' + temp
    else PadSp := temp;
end;

{
  Pad time element with left zero if necessary
}
function LZero(num: Integer): str10;
var
  temp: str10;
begin
  temp := IntStr(num);
  if num<10 then LZero := '0' + temp
    else LZero := temp;
end;

{
  Compute day-of-week using Harry's modified Zeller
Algorithm
}
function Zeller(month,day,year: Integer): Integer;
begin
  if month<3 then dec(year);
```

```
  Zeller := (Zel[month] + day + year + (year shr 2)) mod
7;
end;

{
  Read time directly from clock chip
  works only on ATs, 286s, 386s; does not work on
standard PCs, XTs
}
procedure GetChipTime;
var
  good: Boolean;
begin
  good := false;
  while not good do begin
    port[$70] := $0A;     { wait for update to finish
(then there's 244 us) }
    if (port[$71] and $80)=0 then begin
      port[$70] := $00;     { get seconds }
      Sec := port[$71];
      port[$70] := $0A;
      if (port[$71] and $80)=0 then begin
        port[$70] := $02;    { get minutes }
        Minu := port[$71];
        port[$70] := $0A;
        if (port[$71] and $80)=0 then begin
          port[$70] := $04;     { get hours }
          Hour := port[$71];
          port[$70] := $0A;
          if (port[$71] and $80)=0 then begin
            port[$70] := $07;     { get day }
            Day := port[$71];
            port[$70] := $0A;
            if (port[$71] and $80)=0 then begin
              port[$70] := $08;     { get month }
              Mon := port[$71];
              port[$70] := $0A;
```

```
                if (port[$71] and $80)=0 then begin
                  port[$70] := $09;     { get year }
                  Year := port[$71];
                  port[$70] := $0A;
                  if (port[$71] and $80)=0 then good :=
true;
                end;
              end;
            end;
          end;
        end;
      end;
  end;
  Sec  := (Sec  shr 4) * 10 + (Sec  and $0F);  { convert
from BCD }
  Minu := (Minu shr 4) * 10 + (Minu and $0F);
  Hour := (Hour shr 4) * 10 + (Hour and $0F);
  Day  := (Day  shr 4) * 10 + (Day  and $0F);
  Mon  := (Mon  shr 4) * 10 + (Mon  and $0F);
  Year := (Year shr 4) * 10 + (Year and $0F);
  WeekDay := Zeller(Mon,Day,Year);
end;

{
  Get a time stamp consisting of the relative day and
the current time
}
procedure GetTimeStamp;
var
  tdow,tday,tmon,tyear,tsec,tminu,thour: str10;
  AmPm: Char;
begin
  GetChipTime;
  tmon  := PadSp(Mon);
  tday  := LZero(Day);
  tyear := LZero(Year);
  tdow  := copy(DayOfWeek,Zeller(Mon,Day,Year)*3+1,3);
```

```
  CurrentDate := tdow+' '+tmon+'/'+tday+'/'+tyear;
  tsec  := LZero(Sec);
  tminu := LZero(Minu);
  if Hour=0 then begin
    thour := '12';
    AmPm := 'A';
  end else if Hour<12 then begin
    thour := PadSp(Hour);
    AmPm := 'A';
  end else if Hour=12 then begin
    thour := '12';
    AmPm := 'P';
  end else begin
    thour := PadSp(Hour-12);
    AmPm := 'P'
  end;
  CurrentTime := thour+':'+tminu+' '+AmPm+'M';
end;

{
  Compute difference in seconds for a calling wait
routine
}
procedure WaitTest;
begin
  GetChipTime;
  NewSec := Sec + 60*Minu + 3600*Hour;
  DeltaSecs := NewSec - SaveSec;
  if DeltaSecs < 0 then DeltaSecs := DeltaSecs + 86400;
end;
```

```
{-------------------------------------------------------

                    BUILD Utility Routines

-------------------------------------------------------}

procedure beep;
begin
  Sound(1000);
  delay(300);
  NoSound;
end;

function sgn(num: FloatType): FloatType;
begin
  if num<0 then sgn := -1
  else if num>0 then sgn := 1
  else sgn := 0;
end;

function upper(text: string): string;
var
  i: Integer;
  temp: string;
begin
  temp := text;
  for i := 1 to length(temp) do temp[i] :=
upcase(temp[i]);
  upper := temp;
end;

function many(Ch: Char;num: Integer): string;
var
  temp: string;
  i: Integer;
begin
  temp := '';
```

```
  for i:=1 to num do temp  := temp+Ch;
  many := temp;
end;

function min(int1,int2: Integer): Integer;
begin
  if int1<int2 then min := int1
    else min := int2;
end;

function max(int1,int2: Integer): Integer;
begin
  if int1>int2 then max := int1
    else max := int2;
end;

function IntStr(num: Integer): str16;
var
  temp: str16;
begin
  if num>=0 then Str(num,temp) else Str(num+$8000,temp);

  IntStr := temp;
end;

function WordStr(num: Word): str16;
var
  temp: str16;
begin
  Str(num,temp);
  WordStr := temp;
end;

function LongStr(num: LongInt): str16;
var
  temp: str16;
begin
```

```
  Str(num,temp);
  LongStr := temp;
end;

function RealStr(num: FloatType;AfterDecimal: Integer):
str80;
const
  RoundTable: array[0..6] of Real =
    (0.49,0.049,0.0049,0.00049,0.000049,0.0000049,
0.00000049);
var
  i: Integer;
  temp: str80;
begin
  Str(num+RoundTable[AfterDecimal]:10:AfterDecimal,
temp);
  i := 1;
  while (i<length(temp)) and (temp[i]=' ') do inc(i);
  RealStr := copy(temp,i,99);
end;

function spc(len: Integer): string;
begin
  spc := copy(sp80+sp80,1,len);
end;

{
  convert number to hex byte string
}
function HexByte(num: Byte): str4;
const
  HexChars: str16 = '0123456789ABCDEF';
begin
  HexByte := HexChars[num shr 4 + 1] + HexChars[num and
$0F + 1];
end;
```

```
{
  read the DOS command line
}
procedure ReadCommandLine(var line: string);
var
  i: Integer;
begin
  line := ParamStr(1);
  for i:=2 to ParamCount do
    line := line + ' ' + ParamStr(i);
end;

{
  Remove excess quote marks around string
}
function NoQuotes(Text: str80): str80;
const
  Quote = '"';
var
  s,e: Integer;
begin
  s := 1;
  if (length(Text)>1) and (Text[1]=Quote) then inc(s);
  e := length(Text);
  if Text[e]=Quote then dec(e);
  NoQuotes := copy(Text,s,e-s+1);
end;

{
  Removes all double spaces in text string
}
procedure RemoveDoubleSpaces(var text: string);
begin
  while pos('  ',text)>0 do
    delete(text,pos('  ',text),1);
end;
```

```
{
  Find any char of Chars in Main starting at index Start

  Returns length(Main)+1 if no character of Chars in
Main
}
function FindStr(Start: Integer;Main: string;Chars:
str10): Integer;
var
  i,j,min: Integer;
  temp: String;
begin
  min := length(Main)+1;
  temp := copy(Main,Start,199);
  for i:=1 to length(Chars) do begin
    j := Start + pos(Chars[i],temp) - 1;
    if (j >= Start) and (j < min) then min := j;
  end;
  FindStr := min;
end;

{
  better val routines
}
function IntVal(line: String;var code: Integer):
Integer;
var
  code2: Integer;
  num: Integer;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  IntVal := num;
end;

function WordVal(line: String;var code: Integer): Word;
var
```

```
  code2: Integer;
  num: Word;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  WordVal := num;
end;

function LongVal(line: String;var code: Integer):
LongInt;
var
  code2: Integer;
  num: LongInt;
begin
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  LongVal := num;
end;

function RealVal(line: String;var code: Integer):
FloatType;
var
  code2: Integer;
  num: FloatType;
begin
  if line[1]='.' then line := '0' + line;
  val(line,num,code);
  if code<>0 then val(copy(line,1,code-1),num,code2);
  RealVal := num;
end;

{
  Pick out a number from a string
  starting at Index and going to the next space or comma
or end-of-string
}
```

```
procedure UnitValShortInt(var Storage: Integer;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := IntVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else Index := Index + i;
end;

procedure UnitValInt(var Storage: LongInt;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := LongVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else Index := Index + i;
end;

procedure UnitVal(var Storage: FloatType;Text:
String;var Index: Integer);
var
  i: Integer;
begin
  Storage := RealVal(copy(Text,Index,99),i);
  if i=0 then Index := length(Text)+1
    else Index := Index + i;
end;

{
  Pick out text from a string
  starting at Index and going to the next comma or
end-of-string
}
procedure UnitStr(var Storage,Text: String;var Index:
Integer);
```

```
var
  i: Integer;
begin
  i := FindStr(Index,Text,',');
  Storage := copy(Text,Index,i-Index);
  Index := i + 1;
end;

procedure GetBuildKey; forward;

{
  Show error and store it on disk
}
procedure SystemError(Text: string);
begin
  if OpMode = TextOp then writeln(Text)
    else begin
      MoveTo(3*8,24*14-2);
      OverText(Text,50);
      GetBuildKey;
      TextMode(C80);
    end;
  assign(ErrorFile,ErrorFileName);
  rewrite(ErrorFile);
  writeln(ErrorFIle,'BUILD: ',Text);
  close(ErrorFile);
end;

{
  Can't open file.  Report this to operator.
}
procedure CantOpen(FileName: string);
begin
  SystemError('Can''t open file '+FileName);
end;

{
```

```
  IEEE / Turbo Pascal floating point number conversions
}
procedure ConvertRtoS(r,s: FpPointerType);
  { Converts a Real type (6 byte) number to a Single (4
byte) }
var
  e: Byte;
begin
  e:=r^[0];
  if e<>0 then Dec(e,2);
  s^[3]:=(r^[5] and $80) or ((e shr 1) and $7F);
  s^[2]:=((e shl 7) or (r^[5] and $7F)) and $FF;
  s^[1]:=r^[4];
  s^[0]:=r^[3];
end;

procedure ConvertStoR(s,r: FpPointerType);
  { Converts a Single type (4 byte) number to a Real (6
byte) }
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;
```

```
{-------------------------------------------------------

              BUILD Z-Stage Elevator Routines

-------------------------------------------------------}

procedure InitZTable;
begin
  port[ZAddress+1] := $64;
  port[ZAddress+1] := $60;
  delay(100);
  port[ZAddress+1] := $40;
  port[ZAddress+1] := $60;
  delay(100);
end;

procedure ZTable(Ch: Char);
var
  b: Integer;
begin
  repeat
    b := port[ZAddress+1];
  until (b and $10)>0;
  port[ZAddress] := ord(Ch);
  port[ZAddress+1] := $70;
  repeat
    B:=port[ZAddress+1];
  until (b and $10)=0;
  port[ZAddress+1] := $60;
end;

{
  Send command to Z-axis stepper controller
}
procedure ZMove(Cmd: string);
begin
  if ShowStuff then begin
```

```
    if whereX>1 then writeln;
    writeln('   ZTable>',Cmd);
  end;
  if not ZTableOff then begin
    Cmd := Cmd + Cr;
    for i:=1 to length(Cmd) do ZTable(Cmd[i]);
  end;
end;

procedure FormatZCommand(Distance: FloatType);
begin
  Distance := sgn(Distance) * int(abs(Distance) * ZPitch
* 1000 / 25.4);
  if NegZStage then Distance := -Distance;
  ZMove('D'+RealStr(Distance,0)+' G');
end;

procedure WaitForIdleZStage;
var
  b: Word;
begin
  repeat
    b := port[ZAddress+1];
  until keypressed or (b and 2>0);
end;
```

```
{-------------------------------------------------------

                General Graphics Routines

-------------------------------------------------------}

procedure DrawBorder;
{ Draw a border around the current view port }
begin
  SetLineStyle(SolidLn,0,ThickWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
end;

procedure BuildImage;
begin
  OnScreen := 1-OnScreen;
  SetActivePage(OnScreen);
end;

procedure OverText(Text: String;len: Integer);
var
  x,y,color: Integer;
begin
  x := GetX;
  y := GetY;
  color := GetColor;
  SetFillStyle(EmptyFill,0);
  Bar(x,y,x + len * TextWidth('A'),y + TextHeight(Text)
+ 1);
  SetColor(color);
  OutTextXY(x,y,Text);
end;

procedure ShowImage;
begin
```

```
  SetVisualPage(OnScreen);
end;

function GetInput(maxlen: Integer): String;
var
  key: Char;
  temp: String;
  input: Boolean;
  baseX,baseY,index: Integer;
begin
  temp := '';
  index := 0;
  baseX := getX;
  baseY := getY;
  key := chr(0);
  SetVisualPage(OnScreen);
  while key<>Cr do begin
    MoveTo(baseX+index*8,baseY);
    SetColor(3);
    OutText(chr(219));
    key := ReadKey;
    if key = Ctrl C then Abort;
    MoveTo(baseX+index*8,baseY);
    SetColor(0);
    OutText(chr(219));
    MoveTo(baseX+index*8,baseY);
    SetColor(3);
    if key in [Bs,Cr,' '..'~'] then begin
      if key=Bs then begin
        if index>0 then dec(index);
      end else if key in[' '..'~'] then begin
        OutText(key);
        if index<maxlen then inc(index);
        temp[index] := key;
      end;
    end;
  end;
```

```
  temp[0] := chr(index);
  GetInput := temp;
  SetColor(3);
end;

procedure GrAbort(Msg: string);
begin
  Writeln(Msg,': ',GraphErrorMsg(GraphResult));
  Halt(1);
end;

procedure InstallGraphics;
begin
  { Register all the drivers }
  if RegisterBGIdriver(@CGADriverProc) < 0 then
    GrAbort('CGA');
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    GrAbort('EGA/VGA');

  { Register all the fonts }
  if RegisterBGIfont(@SansSerifFontProc) < 0 then
    GrAbort('SansSerif');
  if RegisterBGIfont(@SmallFontProc) < 0 then
    GrAbort('Small');
  if RegisterBGIfont(@TriplexFontProc) < 0 then
    GrAbort('Triplex');

  GraphDriver := Detect;                { autodetect the
hardware }
  InitGraph(GraphDriver,GraphMode,'');  { activate
graphics }
  if GraphResult<>grOk then             { any errors? }
  begin
    Writeln('Graphics init error: ',
GraphErrorMsg(GraphDriver));
    Halt(1);
  end;
```

```
  SetVisualPage(1);
  ClearDevice;
  OnScreen := 1;  { start with Screen 0 }
end;

{
  Print action information
}
procedure Action(Text: String);
var
  i: Integer;
begin
  if OpMode=TextOp then write(Text,' ')
  else begin
    MoveTo(40*8,7*14);
    SetColor(2);
    OverText(Text,20);
    SetColor(3);
  end;
end;

procedure Message(Text: String);
begin
  MoveTo(3*8,24*14-2);
  OverText(Text,length(Text)+5);
end;
^Z
```

```
{=========================================================
   3D Systems Stereolithography System Software
=========================================================}

   BLDSWEEP.INC
   SLA-1 Shell to run sweeper arm software

-----------------------------------------------------------
    8/19/88  Ver 3.30  Started
-----------------------------------------------------------

{ $ R-,S-,I+,D-,T-,F-,V-,B-,N-,L+ }
{ $ M 16384,0,655360 }

const
  MaxSStep = 1154;
{  LayerDepth = 0.508;  20*(25.4)*(1/1000) mils = .508
mm  }
  LnSteps: array[1 .5] of Integer = (20,9,4,2,1);
  PowerPort = $304
  _ZStage_Out = $02;

var
   FinalLevel : integer;

procedure RotateStep(DirStep: Integer;var InputByte:
Integer);
const
  Bits: array[0..3] of Byte = ($01,$03,$02,$00);
begin
 StepIter  := ((StepIter + DirStep) + 4) mod 4;
 InputByte := Bits[StepIter];
end;


procedure MoveBlade(option:char;Velocity:real);
const
```

```
  Vsubtract = 20;
var
  delaytime : integer;

procedure SweeperBack;
var
  i,j : integer;
begin
   port[$309]:=$FF;
   i := 0;
   if SHOW then begin
      writeln('SWEEPER BACK -           ');
   end;
   for j := 1 to MaxSStep do begin
      if CurrBladePos < MaxSStep then begin
         if i <= 4 then inc(i);
         rotatestep(+1,Sweepstep);
         port[$30A] := (port[$30A] and $F3) or
(sweepstep*4 and $0C);
         inc(CurrBladePos);
         delay(trunc(Velocity) + LnSteps[i]);
      end;
   end;
   port[$309]:=0;
end;

procedure SweeperForward;
var
   i,j : integer;
begin
   port[$309]:=$FF;
   i := 0;
   if SHOW then begin
      writeln('SWEEPER FORWARD -      ');
   end;
   for j := 1 to MaxSStep do begin
      if (port[$30A] and 16) <> 16 then begin
```

```
            CurrBladePos := 0;
            port[$309] := 0;
            exit;  { home limit switch }
         end;
         if i <= 4 then inc(i);
         rotatestep(-1,sweepstep);
         port[$30A] := (port[$30A] and $F3) or (sweepstep*4
and $0C);
         dec(CurrBladePos);
         delay(trunc(velocity) + LnSteps[i]);
      end;
      port[$309]:=0;
end;

begin { main procedure moveblade }
      if (port[$30A] and 16) <> 16 then begin
         CurrBladePos := 0;
         if option = 'H' then exit;  { home limit switch }
      end;
      if CurrBladePos = 0 then SweeperBack else
SweeperForward;
end; { main procedure moveblade }

procedure InitSweeperandPlunger;
begin
      port[$30B] := $98;   { init 8255 }
      port[$30B] := $98;
      port[$30A] := 0;
      port[$309] := 0;

      {  JB is $300, $308 is address of 3rd 8255 chip }

      if SHOW then writeln('Moving sweeper to home
position...');
      MoveBlade('H',Velocity[1]);
      MoveBlade('H',Velocity[1]);
      if SHOW then begin
```

```
      writeln('SWEEPER AND PLUNGER INITIALIZED
        ');
   end;
end;

{-------------------------------------------------------

procedure SetPlunger(numSetPlungers:integer);
var
  level,oldlevel, timesthru : integer;
  OKfirstTry        : boolean;

procedure PlungerDown;
var
  i,j : integer;
begin
{   if SHOW then begin
      writeln('PLUNGER DOWN - setting level     ');
   end; }
   for j := 1 to 15 do begin
      rotatestep(+1,plungestep);
      port[$30A] := (port[$30A] and $FC) or (plungestep
and $03);
      delay(PlungeMotorDelay);
   end;
end;

procedure PlungerUp;
var
 i,j : integer;
begin
 {   if SHOW then begin
      writeln('PLUNGER UP - setting level           ');
   end; }
   for j := 1 to 15 do begin
         rotatestep(-1,plungestep);
```

```
            port[$30A] :=( port[$30A] and $FC) or
(plungestep and $03);
            delay(PlungeMotorDelay);
      end;
end;

procedure ReadBiCell(recurs:char;var newLevel:integer);
const
  LevelReading = 128;
var
  b,i,j : integer;
begin
   port[780] := 5;      { addr ADC port = 780 has 16
channels, use 5 }
   for i := 1 to 100 do inc(j); { 10 us }
   i := 0;
   repeat
      inc(i);
      b := port[$310];
      newLevel := port[$30C];
   until (b<128) or (i>2000); {re-repeat this loop}
   NewLevel := NewLevel - LevelReading;
   FinalLevel := NewLevel;
   if SHOW then begin
  {    writeln('   NewLevel =',NewLevel,'        '); }
      delay(200);
   end;
   if (NewLevel = BIAS) and (Recurs = 'R') then begin
      { test for laser blocked or off bi-cell }
      for i := 1 to 5 do PlungerUp;
      ReadBiCell('N',NewLevel);
      if NewLevel = BIAS then begin
         for i := 1 to 10 do PlungerDown;
         ReadBiCell('N',NewLevel);
         if NewLevel = BIAS then begin
            writeln;
```

```
            writeln(' ERROR : Laser Beam is blocked or
off Bi-Cell.');
            writeln;
            UserWait;
            Halt(1);
         end;
      end;
   end; { NewLevel = BIAS }
end;    { ReadBiCell      }

begin { SetPlunger }
      OKfirstTry := false;
      TimesThru := 0;
      if SHOW then begin
         if NumSetPlungers = -1 then writeln('
AUTOMATIC MODE ')
         else writeln('     Number of Level
Checks=',numSetPlungers);
      end;
      if NumSetPlungers = -1 then NumSetPlungers := 25;
   repeat
      if SHOW then writeln('     READING BI-CELL     ');
      ReadBiCell('R',level);
      if (NumSetPlungers = 25) and (TimesThru>=1) and
         (Level<=UpLim) and (Level>=LowLim) then
OKfirstTry:= true;

    if TimesThru = 0 then begin
      { do a validity test here -- }
      oldlevel := level;
      for i := 1 to 5 do PlungerUp;
      ReadBiCell('R',level);
      if level = oldlevel then begin
         for i := 1 to 10 do PlungerDown;
         ReadBiCell('R',level);
         if level = oldlevel then begin
            writeln;
```

```
            writeln(' ERROR : Laser Beam is not within a
usable bi-cell range.');
            writeln;
            UserWait;
            Halt(1);
         end;
      end else for i := 1 to 5 do PlungerDown;
   end;  { TimesThru = 0 }

      if level > UpLim then begin
         repeat
            PlungerUp;
            ReadBiCell('R',level);
         until level < UpLim;
      end;
      if level < LowLim then begin
         repeat
            PlungerDown;
            ReadBiCell('R',level);
         until level > LowLim;
      end;
      inc(TimesThru);
      if not((TimesThru > NumSetPlungers) or OKfirstTry)
      then delay(PostDipDelay*1000);
   until (TimesThru > NumSetPlungers) or OKfirstTry;
   if SHOW then writeln('   FinalLevel=',FinalLevel);
end;  { SetPlunger }

procedure SweepIt(index:integer);
begin
   if SHOW then begin
      writeln;
      writeln('***  TOP OF SWEEP  sweepnum =',index);
   end;
   if index = 1 then begin
      FormatZCommand(-ZD - BladeGapL0 +
BladeGap[index]);
```

```
      delay(100);
      WaitForIdleZStage;
   end;
   { else FormatZCommand(BladeGap[index] -
BladeGap[index-1]);
   delay(100);
   WaitForIdleZStage; }
   MoveBlade('S',Velocity[index]);
end;

{ ---------------- MAIN PROGRAM ----------------- }

procedure DoSweep;
begin
  ZW := RelaxationTime;              { seconds       }
  ZD := DipDepth;                    { already in mm }
  ZV := ZtableVeloc;
  ZA := ZtableAccel;

  LayerDepth := Thickness;  { HERE is the questionable
portion }

  if SHOW then writeln('   Layer
Thickness(mm)=',LayerDepth:5:3,
                                        ' (mils)=',
(LayerDepth*1000/25.4):5:3);
  for isweep := 1 to 7 do BladeGap[isweep] :=
LayerDepth;
  ZTableOff := false;
  FormatZCommand(ZD);                  { Move Z dip }
  delay(100);
  WaitForIdleZStage;
  delay(postDipDelay*1000);

  if numSweeps <= 0 then begin
     if SHOW then writeln('*** NumSweeps = ',Numsweeps);
```

```
      if (NumSweeps = 0) and (numChecks <> 0) then
SetPlunger(numChecks);
      FormatZCommand(LayerDepth-ZD);
      delay(100);
      WaitForIdleZStage;
      Delay(ZW{RelaxationTime});
      exit;
  end;
  SetPlunger(numChecks);
  for isweep := 1 to numSweeps do SweepIt(isweep);
  FormatZCommand(BladeGapL0-BladeGap[numSweeps]+
LayerDepth);
       { bring zstage back to pre-sweep plus part layer
depth }
  delay(100);
  WaitForIdleZStage;
  OldDeltaSecs := 1;
  {Delay(ZW  RelaxationTime );}
  ZTableOff := true;
end;
```

```
{-------------------------------------------------------

                Laser Controller Routines

-------------------------------------------------------}

{
  Check for Memory-Resident Laser Controller System
}
function LaserOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or LASERVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,BX-19);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      LaserOk := true;
      with regs do
        CopyPtr := Ptr(ES,BX-46);
      move(CopyPtr^[1],LaserVersion[1],4);
      LaserVersion[0] := #4;
    end else begin
      LaserOk := false;
      LaserVersion := '?.??';
    end;
  end;

function LaserPresent: Boolean;
  begin
```

```
    if LaserOk then LaserPresent := true
    else begin
      LaserPresent := false;
      SystemError('LASER system not installed.');
    end;
  end;

{
  Check for Memory-Resident STEREO Stereolithography
Device Driver
}
function StereoOk: Boolean;
  type
    CopyType  = array[1..20] of char;
  var
    CopyPtr: ^CopyType;
    CopyStr: string[20];
  begin
    with regs do begin
      AX := ($35 shl 8) or STEREOVECTOR;
      Intr($21,regs);
      CopyPtr := Ptr(ES,57);
    end;
    move(CopyPtr^[1],CopyStr[1],18);
    CopyStr[0] := #18;
    if CopyStr = 'Tarnz Technologies' then begin
      StereoOk := true;
      with regs do
        CopyPtr := Ptr(ES,30);
      move(CopyPtr^[1],StereoVersion[1],4);
      StereoVersion[0] := #4;
    end else begin
      StereoOk := false;
      StereoVersion := '?.??';
    end;
  end;
```

```
function StereoPresent: Boolean;
  begin
    if StereoOk then StereoPresent := true
    else begin
      StereoPresent := false;
      SystemError('STEREO driver not installed.');
    end;
  end;

procedure ShowError(Error: Integer);
begin
  case Error of
    0: ;
    1: PrintError('1. Laser interface card not found');
    2: PrintError('2. Laser channel not open');
    3: PrintError('3. Background Command Queue full
timeout');
    4: PrintError('4. Invalid interrupt time');
    5: PrintError('5. Not in block mode');
    6: PrintError('6. Exec Queue limit timeout');
    7: PrintError('127. Bad SIH command parameter');
    8: PrintError('128. Unrecognized STEREO command');
    255: PrintError('255. Internal error (please
report)');
  else PrintError(IntStr(Error)+'. Unrecognized SIH
error');
  end;
end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          Pass Command on to Memory Resident STEREO
Device Driver

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - }

function STEREO(LaserCommand: String): Integer;
```

```
begin
  STEREO := 0;
  if length(LaserCommand)>0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      writeln('   Laser>',LaserCommand);
    end;
    if not LaserOff then begin
      if LaserCommand[1]<>'!' then with regs do begin
        LaserCommand := LaserCommand + chr(0);
        AX := $00;   { ASCII text format }
        BX := Ofs(LaserCommand)+1;   { pass pointer to
string }
        ES := Seg(LaserCommand);
        Intr(STEREOVECTOR,regs);
        STEREO := AX and $FF;
        QSize := CX;
      end;
    end;
  end;
  if keypressed then HandleKeyboardRequest;
end;

function BinarySTEREO(CommandBuffer: Pointer): Integer;
var
  BufSeg,BufOfs: Word;
begin
  BinarySTEREO := 0;
  BufSeg := seg(CommandBuffer^);
  BufOfs := ofs(CommandBuffer^);
  if Mem[BufSeg:BufOfs] > 0 then begin
    if ShowStuff then begin
      if whereX>1 then writeln;
      write('   BinaryLaser>');
{}      write('Seg:Ofs=',BufSeg,':',BufOfs,'<');
      for i:=0 to 11 do
write(HexByte(Mem[BufSeg:BufOfs+i]),' ');
```

```
        writeln;
      end;
      if not LaserOff then begin
        with regs do begin
          AX := $01;   { binary format }
          BX := BufOfs;   { pass pointer to binary command
}
          ES := BufSeg;
          Intr(STEREOVECTOR,regs);
          BinarySTEREO := AX and $FF;
          QSize := CX;
        end;
      end;
    end;
    if keypressed then HandleKeyboardRequest;
  end;
```

```
{-------------------------------------------------------

                    NEWVIEW Include File

-------------------------------------------------------}

{$F+} procedure MainExit; {$F-}
  begin
    {$I-} Close(SliceFile); Close(TriangleFile); {$I+}
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrScr;
    ExitProc:=ExitSave;
  end;

procedure SelectTxtColor(select : Integer);
  { Sets text color requested or White for non-color
displays }
  begin
    if MonoDisplay then TextColor(White)
    else TextColor(select);
  end;


procedure SelectGraphColor(select : Integer);
  { Sets graphic color.  If black and white, uses dashed
lines, etc. }
  var
    LineType : Integer
  begin
    if MonoDisplay then begin
      SetColor(White);
      case select of
         0 : SetColor(Black);
         1 : LineType:=SolidLn;
         2 : LineType:=CenterLn;
```

```
            4 : begin
                  LineType:=CenterLn;
                  SetColor(LightGray);
                end;
            9 : begin
                  LineType:=SolidLn;
                  SetColor(LightGray);
                end;
           10 : LineType:=DashedLn;
           12 : begin
                  LineType:=DashedLn;
                  SetColor(LightGray);
                end;
          ELSE LineType:=CenterLn;
        end;
        SetLineStyle(LineType,0,NormWidth);
      end else SetColor(select);
    end;

procedure CursorOff;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
      AH := 1;
      CH := CH or $20;
      Intr($10,regs);
   end;
end;

procedure CursorOn;
begin
   with regs do begin
      AH := 3;
      BH := 0;
      Intr($10,regs);
```

```
      AH := 1;
      CH := CH and $DF;
      Intr($10,regs);
    end;
end;


{*******************************************************
procedure ConvertRtoS(r,s : PointerType);
  { Converts a Real type (6 byte) number to a Single (4
byte)
var
  e : Byte;
begin
  e:=r^[0];
  if e<>0 then Dec(e,2);
  s^[3]:=(r^[5] and $80) or ((e shr 1) and $7F);
  s^[2]:=((e shl 7) or (r^[5] and $7F)) and $FF;
  s^[1]:=r^[4];
  s^[0]:=r^[3];
end;
*******************************************************}

procedure ProgramError(ErrorLine : String);
  { Print Program Error and exits program }
  begin
    RestoreCrtMode;
    TextBackground(Black);
    TextColor(White);
    ClrScr;
    writeln('*** SLAGRAPH ERROR: ',ErrorLine,' ***');
    ch:=ReadKey;
    Halt(1);
  end;

procedure ReAllocate(ch : Char;Alloc : LongInt);
  { Allocates the array space to sugessted number }
```

```
  var
    MemoryAvail : LongInt;
  begin
    { Free Heap }
    Release(HeapTop);
    SliceFilename   := '';
    TriangleFilename:= '';
    MemoryAvail := MemAvail - 32768;

    if ch='S' then begin
      { Don't allow more than machine can handle }
      if Alloc>MAX_LAYERS then Total_Layers:=MAX_LAYERS
      else Total_Layers:=Alloc;
      { Make sure you can allocate that many alone. If
not, set maximum }
      if Total_Layers*LAYER_SIZE>MemoryAvail then
Total_Layers:=Round(MemoryAvail/LAYER_SIZE);
      { Given number of Layers, fill rest of memory with
Triangles }
      Total_Triangles:=Round((MemoryAvail-Total_Layers
*LAYER_SIZE)/TRIANGLE_SIZE);
      if Total_Triangles>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES;
    end else begin
      { Don't allow more than machine can handle }
      if Alloc>MAX_TRIANGLES then
Total_Triangles:=MAX_TRIANGLES
      else Total_Triangles:=Alloc;
      { Make sure you can allocate that many alone. If
not, set maximum }
      if Total_Triangles*TRIANGLE_SIZE>MemoryAvail then
Total_Triangles:=Round(MemoryAvail/TRIANGLE_SIZE);
      { Given number of Triangles, fill rest of memory
with Layers }
      Total_Layers:=Round((MemoryAvail-Total_Triangles
*TRIANGLE_SIZE)/LAYER_SIZE);
```

```
      if Total_Layers>MAX_LAYERS then
Total_Layers:=MAX_LAYERS;
    end;
    GetMem(SliceLayerValue,Total_Layers*2);
    GetMem(SliceLayerPtr,Total_Layers*4);
    GetMem(SliceLayerBlocks,Total_Layers*2);
    GetMem(SliceLayerXMin,Total_Layers*2);
    GetMem(SliceLayerXMax,Total_Layers*2);
    GetMem(SliceLayerYMin,Total_Layers*2);
    GetMem(SliceLayerYMax,Total_Layers*2);
    GetMem(TrianglePtr,Total_Triangles*4);
    GetMem(TriangleAxisValue,Total_Triangles*6);
    Dec(Total_Triangles);
    Dec(Total_Layers);
  end;


procedure OutTextLn(aline:String);
  { "writeln" for graphics }
  begin
    OutText(aline);
    MoveTo(1,GetY+TextHeight(' ')+2);
  end;

procedure GrTextXY(aline:String;x,y : Byte);
  { writes text at XY char on graphic screen }
  begin
    OutTextXY((x-1)*(TextWidth('
')+2),(y-1)*(TextHeight(' ')+2),aline);
  end;

procedure ConvertStoR(s,r : PointerType);
  { Converts a Single type (4 byte) number to a Real (6
byte) }
begin
  r^[0]:=(((s^[3] shl 8) + s^[2]) shr 7) and $FF;
  r^[1]:=0;
```

```
  r^[2]:=0;
  r^[3]:=s^[0];
  r^[4]:=s^[1];
  r^[5]:=(s^[3] and $80) or (s^[2] and $7F);
  if r^[0]<>0 then Inc(r^[0],2); { e offset }
end;

procedure ToUpper(var aline : String);
  { Converts a string to upper case }
  var
    i : Integer;
  begin
    for i:=1 to Length(aline) do aline[i] :=
UpCase(aline[i]);
  end;

procedure Center(str : String;linenum : Byte);
  { Centers text on screen line }
  begin
    GotoXY(1,linenum);
    ClrEol;
    if Length(str)>79 then GotoXY(1,linenum)
    else GotoXY(40-(Length(str) div 2),linenum);
    write(str);
  end;

procedure WaitKey;
  { Waits for a key press while prompting }
  var
    ch : Char;
  begin
    Center('<Press any key to continue>',25);
    ch:=ReadKey;
  end;

procedure PrintHeader;
  { Prints general 3D header at top of screen }
```

```
  begin
    TextColor(White);
    if Not MonoDisplay then TextBackground(Blue)
    else TextBackground(Black);
    ClrScr;
    Center('3D Systems Graphics Display Program (Version
'+VERSION+')',1);
    Center('Copyright (C) 1988 3D Systems Inc',2);
    SelectTxtColor(LightGreen);
  end;

procedure PrintSliceHeader;
  { Print Slice Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Slice File Control Functions',3);
    TextColor(White);
  end;

procedure PrintTriHeader;
  { Print Triangle Screens header }
  begin
    PrintHeader;
    SelectTxtColor(LightRed);
    Center('Triangle File Control Functions',3);
    TextColor(White);
  end;

function GetFilename(var FileName : String; sel :
Integer) : Boolean;
  { Displays file with SLI or STL extentions and allows
user to select
     a file to use. }
  var
    DirRec          : SearchRec;
    CurrFileName    : String;
```

```
    Extn            : String;
    row,col         : Integer;
    count           : Integer;
  begin
    DisplayMode := 'W';
    WSorted  := true; { because when we get the file it
will be wireframe }
    SSorted  := false;
    FileError:=FALSE;
    if sel = 1 then begin
      Extn := '.STL';
      PrintTriHeader;
      GotoXY(3,6);
      write('Triangle');
    end else begin
      Extn := '.SLI';
      PrintSliceHeader;
      GotoXY(3,6);
      write('Slice');
    end;
    { Display the entries }
    write(' Files in current directory:');
    row := 0;
    col := 0;
    count := 0;
    FindFirst('*'+Extn,AnyFile,DirRec);
    while (DosError = 0) and (count < 75) do begin
      GotoXY(3+col*15,8+row);
      write(Copy(DirRec.name,1,Pos('.',DirRec.name)-1));
      col := col + 1;
      if col = 5 then begin
        col := 0;
        row := row + 1;
      end;
      FindNext(DirRec);
    end;
    { Read the filename }
```

```
    repeat
      GotoXY(1,24);
      ClrEol;
      write('   Enter File Name: ');
      Readln(CurrFilename);
      ToUpper(CurrFilename);
      DosError:=0;
      if CurrFilename <> '' then begin
        { Find in directory }
        if Pos('.',CurrFilename) = 0 then CurrFilename
:= CurrFilename + Extn;
        if Pos('*',CurrFilename)=0 then begin
          FindFirst(CurrFilename,AnyFile,DirRec);
          if DosError <> 0 then Center('****
"'+CurrFilename+'" File Not Found ****',25);
        end else begin
          DosError:=1;
          Center('*** No Wild cards allowed ***',25);
        end;
        if DosError<>0 then CurrFilename:='';
      end;
    until (DosError = 0);
    if CurrFilename = '' then GetFilename:=FALSE
    else begin
      FileName := CurrFilename;
      GetFilename:=TRUE;
    end;
   end;

procedure ASCError(FileName : String);
  { Prints ASC file error }
  begin
    Str(StartLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
byte offset '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
```

```
  end;

procedure ASCErrorLn(FileName : String);
  { Prints ASC file error }
  begin
    Str(CurrLinePos,TempStr);
    Center('Error found in Text file "'+FileName+'" at
line '+TempStr,23);
    Center('Line: '+Copy(ErrLine,1,73),24);
    WaitKey;
  end;

procedure CheckBuffer(var InFile : File);
  { Makes sure the input buffer has enogh data in it.
     If not, it reads more in. }
  var
    NumRead : Word;
  begin
    if InBufPtr > InBufEnd-160 then begin
      { Reload Buffer }
      if InBufPtr > InBufEnd then InBufEnd := 0  {
Buffer Empty }
      else begin
        { Buffer almost empty }
         InBufEnd := InBufEnd - InBufPtr + 1;
         Move(InBuffer[InBufPtr],InBuffer[0],InBufEnd);
      end;
      { Fill rest of buffer }
      if Not Eof(InFile) then
BlockRead(InFile,InBuffer[InBufEnd],BUF_SIZE-InBufEnd,Nu
mRead)
      else NumRead:=0;
      { Set buffer pointers }
      if InBufEnd+NumRead=0 then EndOfFile:=TRUE
      else Inc(InBufEnd,NumRead - 1);
      InBufPtr := 0;
    end;
```

```
  end;

procedure ResetFile(var InFile : File);
  { Resets input file to top of file }
  begin
    Reset(InFile,1);
    Seek(InFile,0);
    CurrFilePos:=0;
    CurrLinePos:=0;
    InBufPtr := BUF_SIZE+1;
    InBufEnd := BUF_SIZE;
    EndOfFile:=FALSE;
    FileError:=FALSE;
  end;

procedure PositionFile(var InFile : File;charnum :
LongInt);
  { Positions input buffer to specific character of
file.  If not in buffer
      reads from file }
  begin
    if (charnum>=CurrFilePos-InBufPtr) AND
(charnum<=CurrFilePos-InBufPtr+InBufEnd) then
      InBufPtr:=InBufPtr+charnum-CurrFilePos
    else begin
      Seek(InFile,charnum);
      InBufPtr := BUF_SIZE+1;
      InBufEnd := BUF_SIZE;
      EndOfFile:=FALSE;
    end;
    CurrFilePos:=charnum;
    CheckBuffer(InFile);
  end;

procedure ReadLine(var InFile : File);
  { Reads a line from input file into a string }
  var
```

```
    EOFPos,CRPos,i        : Integer;
  begin
    InputLine:='';
    if Not (FileError or EndOfFile) then repeat
      CheckBuffer(InFile);
      { Search for CR }
      if Not EndOfFile then begin
        if InBufEnd=0 then i:=0
        else i := InBufEnd-InBufPtr+1;
        if i > 255 then i := 255;
        InputLine[0] := Char (i);
        Move(InBuffer[InBufPtr],InputLine[1],i);
        CRPos := Pos(CR,InputLine)-1;
        if CRPos < 0 then CRPos := i;
        InputLine[0] := Char (CRPos);
        EOFPos := Pos(EOF_CHAR,InputLine)-1;
        if EOFPos>0 then InputLine[0] := Char (EOFPos);
        StartLinePos:=CurrFilePos;
        PositionFile(InFile,CurrFilePos+CRPos+1);
        if InBuffer[InBufPtr] = LF then
PositionFile(InFile,CurrFilePos+1);
        Inc(CurrLinePos);
        ToUpper(InputLine);
        ErrLine:=InputLine;
      end else InputLine := '';
    until EndOfFile OR (InputLine <> '');
    EndOfFile := EndOfFile and (InputLine='');
  end;

function ParseLine(var aline : String) : String;
  { Parses first word from string and truncs off from
original string }
  var
    parsed : String;
    i,j    : Integer;
  begin
    i:=1;
```

```
    while (i<=Length(aline)) AND (aline[i]=' ') do
Inc(i);
    if i>Length(aline) then begin
      aline:='';
      parsed:='';
    end else begin
      j:=i;
      while (j<=Length(aline)) AND (aline[j]<>' ') do
Inc(j);
      Move(aline[i],Parsed[1],j-i);
      parsed[0]:=Chr (j-i);
      if j>=Length(aline) then aline:=''
      else begin
        Move(aline[j+1],aline[1],Length(aline)-j);
        aline[0]:=Char (Length(aline)-j);
      end;
    end;
    ParseLine:=parsed;
  end;

procedure Switch(First,Second : LongInt) ;
begin
  TempReal := TriangleAxisValue^[First];
  TriangleAxisValue^[First]:=TriangleAxisValue^[Second];

  TriangleAxisValue^[Second]:=TempReal;
  TempLong := TrianglePtr^[First];
  TrianglePtr^[First]:=TrianglePtr^[Second];
  TrianglePtr^[Second]:=TempLong;
end;


procedure QuickSort(Left,Right : Word);
  var
    LeftIndex,RightIndex : LongInt;
    KeyValue : Real;
begin
```

```
    LeftIndex  := Left ;
    RightIndex := Right ;
    KeyValue := TriangleAxisValue^[(Left+Right) DIV 2] ;

    repeat
      while TriangleAxisValue^[LeftIndex]<KeyValue do
Inc(LeftIndex);
      while KeyValue<TriangleAxisValue^[RightIndex] do
Dec(RightIndex);
      if LeftIndex <= RightIndex then begin
        Switch(LeftIndex, RightIndex);
        Inc(LeftIndex);
        Dec(RightIndex);
      end;
    until LeftIndex > RightIndex ;
    if Left<RightIndex then QuickSort(Left, RightIndex);
    if LeftIndex<Right then QuickSort(LeftIndex, Right);
  end;

procedure SetMinMax(var num,min,max : Word);
  { Updates Min and Max values with given data }
  begin
    if num>max then max:=num;
    if num<min then min:=num;
  end;

function GraphDriverName(DriverNum : Integer) : String;
  begin
    Case DriverNum of
      1 : GraphDriverName:='CGA';
      2 : GraphDriverName:='MCGA';
      3 : GraphDriverName:='EGA';
      4 : GraphDriverName:='EGA 64';
      5 : GraphDriverName:='EGA Monochrome';
      6 : GraphDriverName:='Reserved';
      7 : GraphDriverName:='Hercules Monochrome';
      8 : GraphDriverName:='ATT400';
```

```
        9 : GraphDriverName:='VGA';
        10: GraphDriverName:='PC3270';
        else begin
          Str(DriverNum,TempStr);
          GraphDriverName:='Unknown Type '+TempStr;
        end;
      end;
    end;


procedure SetGraph;
  { Finds what type of video hardware is installed and
picks best graphics screen.  Sets screen definition
information. }
  var
    aspectx,aspecty : Word;
  begin
    DetectGraph(GraphDriver,GraphMode);
    if GraphDriver < 0 then ProgramError('Graphics -
'+GraphErrorMsg(GraphDriver));
    if GraphDriver > 5 then ProgramError('Graphics -
'+GraphDriverName(GraphDriver)+' graphics card not
supported');
    if GraphDriver <=2 then
rc:=RegisterBGIdriver(@CGADriverProc)
    else rc:=RegisterBGIdriver(@EGAVGADriverProc);
    if rc<0 then ProgramError('Graphics -
'+GraphDriverName(GraphDriver)+':
'+GraphErrorMsg(GraphResult));
    GetModeRange(GraphDriver,TempInt,GraphMode);
    if GraphMode>3 then GraphMode:=3; { for CGA }
    InitGraph(GraphDriver, GraphMode, 'c:\tp');
    rc := GraphResult;
    if rc <> GrOk then ProgramError('Graphics -
'+GraphErrorMsg(rc));
    if (GraphDriver<5) or (GraphDriver=8) then
MonoDisplay:=FALSE
```

```
    else MonoDisplay:=TRUE;
    GetAspectRatio(aspectx,aspecty);
    AspectRatio := aspectx/aspecty;
    SizeX := GetMaxX + 1;
    SizeY := GetMaxY + 1;
    GraphCenterX:= Round(SizeX/2);
    GraphCenterY:= Round(SizeY/2);
  end;

procedure SetupGraphicW(MinXW,MaxXW,MinYW,MaxYW : Word);
  { Sets up graphic vars using Integer(Word) proposed
ranges }
  var
    scalex : Real;
    MinX,MaxX,MinY,MaxY : Real;
  begin                {here}
    MinX := MinXW;
    MaxX := MaxXW;
    MinY := MinYW;
    MaxY := MaxYW;
    GraphScale:=(SizeX-2)/((MaxX-MinX)/AspectRatio);
    if (MaxY-MinY)*GraphScale>SizeY then
GraphScale:=(SizeY-4)/(MaxY-MinY);
    GrOffsetX:=MinX+(MaxX-MinX)/2;
    GrOffsetY:=MinY+(MaxY-MinY)/2;
    SetGraphMode(GraphMode);
    SetColor(White);
    SetBkColor(Black);
    SetLineStyle(SolidLn,0,NormWidth);
  end;

procedure SetupGraphicR(MinX,MaxX,MinY,MaxY,MinZ,MaxZ :
Real);
  { Sets up graphic vars using Real values and assuming
rotations }
  var
    RangeX,RangeY,RangeZ : Real;
```

```
  begin
    RangeX:=(MaxX-MinX);
    RangeY:=(MaxY-MinY);
    RangeZ:=(MaxZ-MinZ);
    GraphScale :=
SizeY/Sqrt(Sqr(RangeX)+Sqr(RangeY)+Sqr(RangeZ));
    GrOffsetX:=MinX+RangeX/2;
    GrOffsetY:=MinY+RangeY/2;
    GrOffsetZ:=MinZ+RangeZ/2;
    SetGraphMode(GraphMode);
    SetColor(White);
    SetBkColor(Black);
    SetLineStyle(SolidLn,0,NormWidth);
  end;

procedure DrawLine(x1,y1,x2,y2 : Word);
  { Draws a line on graphics screen (no rotation) }
  var
    rx1,ry1,rx2,ry2 : Real;
  begin
    rx1:=x1;
    ry1:=y1;
    rx2:=x2;
    ry2:=y2;
    x1:=GraphCenterX+Round((rx1-GrOffsetX)*GraphScale/
AspectRatio);
    x2:=GraphCenterX+Round((rx2-GrOffsetX)*GraphScale/
AspectRatio);
    y1:=GraphCenterY-Round((ry1-GrOffsetY)*GraphScale);
    y2:=GraphCenterY-Round((ry2-GrOffsetY)*GraphScale);
    Line(x1,y1,x2,y2);
  end;

procedure Vertex3Dzoom(x,y,z : Real;var new_x,new_z :
Integer);
  { Maps a 3D point into 2D graphics screen }
  begin
```

```
    x:=-(x-GrOffsetX)*zoom;
    y:=(GrOffsetY-y)*zoom;
    z:=(z-GrOffsetZ)*zoom;
    new_x := Round(HoriOffset*zoom) +
    (GraphCenterX + Round(GraphScale*(x*Gr3DMat[0,0]+y*
Gr3DMat[1,0]+z*Gr3DMat[2,0])/AspectRatio));

    {  Have Y mapping just for reference.
    new_y := (GraphCenterY - Round(GraphScale*(x*Gr3DMat
[0,1]+y*Gr3DMat[1,1]+z*Gr3DMat[2,1])));
    }

    new_z := Round(-VertOffset*zoom)+
    (GraphCenterY - Round(GraphScale*(x*Gr3DMat[0,2]
+y*Gr3DMat[1,2]+z*Gr3DMat[2,2])));
  end;

procedure Vertex3D(x,y,z : Real;var new_x,new_z :
Integer);
  { Maps a 3D point into 2D graphics screen }
  begin
    x:=-(x-GrOffsetX);
    y:=(GrOffsetY-y);
    z:=z-GrOffsetZ;
    new_x := (GraphCenterX + Round(GraphScale*(x*Gr3DMat
[0,0]+y*Gr3DMat[1,0]+z*Gr3DMat[2,0])/AspectRatio));
   {  Have Y mapping just for reference.
    new_y := (GraphCenterY - Round(GraphScale*(x*
Gr3DMat[0,1]+y*Gr3DMat[1,1]+z*Gr3DMat[2,1])));
    }
    new_z := (GraphCenterY - Round(GraphScale*(x*
Gr3DMat[0,2]+y*Gr3DMat[1,2]+z*Gr3DMat[2,2])));
  end;

procedure Compute3D(x,y,z : Real;var new_x,new_z :
Integer);
```

```
  { Transforms 3D into 2D with no part scaling or
offsetting }
  begin
    new_x := Round((-x*Gr3DMat[0,0]-y*Gr3DMat
[1,0]+z*Gr3DMat[2,0])/AspectRatio);
    new_z :=
Round(-x*Gr3DMat[0,2]-y*Gr3DMat[1,2]+z*Gr3DMat[2,2]);
  end;

procedure DrawAxis;
  { Draws Axis in courner of screen }
  var
    coorx,coory : Integer;
    origx,origy : Integer;
  begin
    origx:=SizeX-35;
    origy:=35;
    SelectGraphColor(4);
    Compute3D(20,0,0,coorx,coory);
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    SelectGraphColor(12);
    Compute3D(0,20,0,coorx,coory);
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    SelectGraphColor(1);
    Compute3D(0,0,20,coorx,coory);
    Line(origx,SizeY-origy,coorx+origx,SizeY-
(coory+origy));
    Compute3D(20,0,0,coorx,coory);
    SelectGraphColor(white);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'X');
    Compute3D(0,20,0,coorx,coory);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'Y');
    Compute3D(0,0,20,coorx,coory);
    OutTextXY(coorx+origx,SizeY-(coory+origy),'Z');
  end;
```

```
procedure DrawPartLine(x1,y1,z1,x2,y2,z2 : Real);
  { Takes 2 3D points and draws line between them }
  var
    coorx1,coory1,coorx2,coory2 : Integer;
  begin
    Vertex3Dzoom(x1,y1,z1,coorx1,coory1);
    Vertex3Dzoom(x2,y2,z2,coorx2,coory2);
    Line(coorx1,coory1,coorx2,coory2);
  end;

procedure Init3D;
  { Sets up 3D transformation matrix for mapping 3D to
2D }
  var
    txy,txz,tyz : Real;
    h2,h1       : Integer;
  begin
    h2  :=  TriangleAngle[2] + DY;   { Vertical Axis }
    h1  :=  TriangleAngle[1] + DX;   { Horizontal Axis }
    if h2>=360 then h2 := h2-360;
    if h1>=360 then h1 := h1-360;
    txy :=  -TriangleAngle[3]*0.0175; { Into Screen}
    txz :=  h2*0.0175; { Vertical Axis }
    tyz :=  h1*0.0175; { Horizontal Axis }

    Gr3DMat[0,0]:= Cos(txz)*Cos(txy);
    Gr3DMat[1,0]:= Cos(tyz)*Sin(txy)-Sin(tyz)*Sin(txz)
*Cos(txy);
    Gr3DMat[2,0]:=-Sin(tyz)*Sin(txy)-Cos(tyz)*Sin(txz)*
Cos(txy);
    Gr3DMat[0,1]:=-Cos(txz)*Sin(txy);
    Gr3DMat[1,1]:= Cos(tyz)*Cos(txy)+Sin(tyz)*Sin(txz)*
Sin(txy);
    Gr3DMat[2,1]:=-Sin(tyz)*Cos(txy)+Cos(tyz)*Sin(txz)*
Sin(txy);
    Gr3DMat[0,2]:= Sin(txz);
```

```
    Gr3DMat[1,2]:= Sin(tyz)*Cos(txz);
    Gr3DMat[2,2]:= Cos(tyz)*Cos(txz);
{    writeln(gftest,'mat',Gr3dmat[0,1],'/',gr3dmat
[1,1],'/',gr3dmat[2,1]);}
  end;


{-------------------------------------------------------
  |       Start of Main Triangle Routines          |
-------------------------------------------------------}

function FindTriangleLevel(value : Real;highlo : Byte) :
Word;
  { Binary search for value and then file lowest or
highest common index }
  var
    low,mid,high,index  :  Word;
  begin
    low:=0;
    high:=TriangleTotal;
    index:=MAX_WORD;
    repeat
      mid:=(low+high) div 2;
      if value<TriangleAxisValue^[mid] then high:=mid
      else if value>TriangleAxisValue^[mid] then
low:=mid
           else index:=mid;
    until (high-low=1) or (index<MAX_WORD);
    if highlo=1 then begin
      if index=MAX_WORD then index:=high;
      while (index>0) and (TriangleAxisValue^[index-1]=
TriangleAxisValue^[index]) do Dec(index)
    end else begin
      if index=MAX_WORD then index:=low;
      while (index<TriangleTotal) and (TriangleAxisValue
^[index+1]=TriangleAxisValue^[index]) do Inc(index);
    end;
```

```
    FindTriangleLevel:=index;
  end;
```

Claims

1. Apparatus for profiling a beam comprising:

means for measuring the intensity of a portion of a predetermined size of the beam when said beam is incident on said means;

means for changing the relative perpendicular displacement of said measuring means from an optical path followed by said beam in order to measure the intensity of same or all portions of the beam along a surface substantially perpendicular to the optical path followed by said beam.

2. Method for profiling a beam, comprising:

the step of measuring the intensity of a portion of preselected size of said beam along a surface substantially perpendicular to an optical path followed by said beam; and

repeating the measuring step for other portions of said beam.

3. The apparatus of claim 1 or 2 wherein said beam is a laser beam.

4. Apparatus for determining the location of a beam

projection on a surface comprising:

sensor means sensitive to the presence of the beam;

scanning means for positioning the beam;

beam profiling means and position readout means for providing information including the apparent location of said sensor means when the beam is pointed at said sensor means.

6. The apparatus of claim 5 wherein said beam is a laser beam.

5. The apparatus of claim 4 wherein said beam is a beam of electromagnetic radiation.

6. The apparatus of claim 4 or 5 wherein said beam is a collimated beam of light.

7. The apparatus of claim 4 wherein said beam is an electron beam.

8. The apparatus of any of the preceding claims 4-7 wherein said sensor means is sensitive to electromagnetic radiation.

9. The apparatus of any of the preceding claims 4-7 wherein said sensor means is sensitive to electron bombardment.

10. The apparatus of any of the preceding claims 4-9 wherein said scanning means for positioning the beam comprises two rotating mirrors under computer control.

11. The apparatus of any of the preceding claims 4-9 wherein said scanning means for positioning the beam comprises two linear positioning motors.

12. A method for determining the location of a beam projection on a surface comprising:

using a sensor means sensitive to the presence of the beam;

scanning means for positioning the beam;

beam profiling means and position readout means for providing information including the apparent location of said sensor means when the beam is pointed at said sensor means.

13. An apparatus for determining the cure depth of a trace of a reacting beam across a photopolymer comprising:

at least one sensor means responsive to the reacting beam;

and beam profiling means.

14. A method for determining the cure depth of a trace of a reacting beam across a photopolymer comprising the steps of:

obtaining a beam intensity profile of the beam;

and computing the cure depth based upon said beam intensity profile.

15. An apparatus for determining the focus of a reacting beam on a surface comprising:

at least one sensor means responsive to the reacting beam;
and beam profiling means.

16. A method for determining the focus of a reacting beam on a surface comprising the steps of:

obtaining a beam intensity profile of the beam;
and computing the focus based upon said beam intensity profile.

17. An apparatus for measuring the power of an incident beam comprising:

at least one sensor means responsive to the beam;
and beam profiling means.

18. A method for measuring the power of an incident beam comprising the steps of:

obtaining a beam intensity profile of the beam;
and computing the power based upon said beam intensity profile.

19. Apparatus for calibrating and normalizing a stereolithographic apparatus for the production of three-dimensional objects from a working medium capable of solidification when exposed to a reaction means operating in a prescribed manner upon a designated surface of the working medium to solidify successive adjacent laminae forming the object, comprising:

(a) at least one sensor means capable of being positioned at a predetermined location in a plane defined by the designated surface, said sensor means capable of sensing when the reaction means is pointed at said sensor means;
(b) positioning means for positioning the reaction means in response to position means information;
(c) memory means for storing position means information corresponding to predetermined locations in a plane defined by the designated surface in response to said sensor means sensing the reaction means pointed at said sensor means.

20. The apparatus of claim 19 further comprising:

(d) processor means for carrying out linear interpolation of said position means information; said processor means capable of providing intermediate position means information for directing said positioning means.

21. A method for calibrating and normalizing a stereolithographic apparatus for the production of three-dimensional objects from a working medium capable of solidification when exposed to a reaction means operating in a prescribed manner upon a designated surface of the working medium to solidify successive adjacent laminae forming the object, comprising the following steps:

(a) positioning at least one sensor means at a predetermined location in a plane defined by the designated surface, said sensor means capable of sensing when the reaction means is pointed at said sensor means;
(b) positioning the reaction means at said sensor means;
(c) recording in memory means position means information corresponding to the location of said sensor means at the time the reaction means was pointed at said sensor means.

22. The method of claim 21 further comprising the following steps:

(d) positioning the reaction means in response to linear interpolations of said position means information corresponding to the location of said sensor means at the time the reaction means was pointed at said sensor means.

23. A method for calibrating and normalizing a stereolithographic apparatus for the production of three-dimensional

objects from a fluid medium capable of solidification when exposed to a reaction means operating in a prescribed manner upon a designated surface of the fluid medium to solidify successive adjacent laminae forming the object, comprising the following steps:

(a) positioning sensors at predetermined locations in a plane corresponding to the designated surface;
(b) scanning the reaction means to be detected by the sensors;
(c) generating information for the position of the reaction means when it is detected by the sensors;
(d) comparing the position information of the reaction means for each of the sensors with the predetermined locations of the sensors; and
(e) generating a table of reaction means position information versus predetermined sensor location for use in positioning the reaction means to construct objects.

24. Apparatus for calibrating an apparatus for the production of three-dimensional parts by application of a reaction beam upon a medium capable of solidification upon contact with the beam comprising:

a calibration plate having a plurality of predetermined paths through which the beam is capable of travelling;
said calibration plate including at least one sensor means for detecting the beam;
said sensor means capable of producing a signal indicative of the beam traveling through a single one of said predetermined paths;
memory means for recording beam position information which corresponds to the beam travelling along a particular one of said predetermined paths.

25. Apparatus for correcting for drift in the production of three-dimensional objects by stereolithography from a fluid medium capable of solidification when exposed to reaction means operating in a prescribed manner upon a designated surface of the fluid medium to provide successive adjacent solid laminae forming the object, comprising:

(a) at least one sensor capable of detecting application of the reaction means and mounted at predetermined locations fixed with respect to the designated surface of the fluid medium;
(b) positioning means for directing the reaction means so that the sensor detects the reaction means and supplies positioning information for the same corresponding to the apparent location of the sensor;
(c) means for comparing the positioning information obtained from the positioning means at a given time to reference position information for directing the reaction means at the sensor taken at an earlier time in order to determine the drift between the positioning information for the sensor taken at the reference and later times; and
(d) means for correcting an information table of positioning in response to the drift.

26. A method for correcting for the drift in stereolithographic apparatus for the production of three-dimensional objects from a fluid medium capable of solidification when exposed to reaction means operating in a prescribed manner upon a designated surface of the fluid medium to solidify successive adjacent solid laminae forming the object, comprising the following steps:

(a) at a reference time, determining positioning information of the reaction means that will cause the reaction means to be detected by one or more sensors mounted at predetermined locations fixed with respect to the designated surface of the fluid medium;
(b) performing step (a) above at later times when drift is to be corrected;
(c) comparing the positioning information from the reaction means at such later time to the reference time to determine drift; and
(d) correcting the positioning information sent to the reaction means in drawing the laminae of the object.

27. Apparatus for correcting for drift errors in the stereolithographic production of a part from a medium capable of solidification upon exposure to reaction means operating in a preselected manner upon a designated surface of the medium to provide adjacent solid laminae forming the part, comprising:

(a) at least one sensor means mounted in a predetermined location fixed with respect to the designated surface of the medium, said sensor means capable of detecting application of the reaction means to said sensor means;
(b) positioning means for applying the reaction means, said positioning means capable of applying the reaction means to said sensor means and said positioning means capable of being adjusted by a drift correction;

(c) current sensor location output means capable of providing current sensor location information, said current sensor location information including the current apparent location of said sensor means;

(d) memory means, said memory means capable of containing past sensor location information, said past sensor location information including a prior apparent location of said sensor means; and

(e) means for comparing said current sensor location information and said past sensor location information including means for determining said drift correction.

28. A method for correcting for drift errors in the stereolithographic production of a part from a medium capable of solidification upon exposure to reaction means operating in a preselected manner upon a designated surface of the medium to provide adjacent solid laminae forming the part, comprising the following steps:

(a) at a reference time, causing the reaction means to be applied to at least one sensor means mounted in a predetermined location fixed with respect to the designated surface of the medium, said sensor means capable of detecting application of the reaction means to said sensor means, and determining and storing in memory means corresponding position information;

(b) at a time later than said reference time performing step (a) above and determining later corresponding position information;

(c) determining a drift error correction by comparing said corresponding position information and said later corresponding position information; and

(d) utilizing said drift error correction to correct the positioning of the reaction means.

29. Apparatus for correcting for drift in the position of a beam on a surface by a beam positioning means, comprising:

(a) at least one sensor means capable of detecting application of the beam to said sensor and mounted at a predetermined location fixed with respect to the surface;

(b) the beam positioning means capable of applying the beam to said sensor means;

(c) the beam positioning means capable of being adjusted by a drift correction;

(d) current sensor location output means capable of providing current sensor location information, said current sensor location information including the current apparent location of said sensor means;

(e) memory means, said memory means capable of containing past sensor location information, said past sensor location information including a prior apparent location of said sensor means; and

(f) means for comparing said current sensor location information and said past sensor location information including means for determining said drift correction.

powered metal capable of being melted by application of the beam.

30. A method for correcting for drift in the positioning of a beam on a surface by a beam positioning means, comprising the following steps:

(a) at a reference time, causing the beam to be applied to at least one sensor means mounted in a predetermined location fixed with respect to the surface, said sensor means capable of detecting application of the beam to said sensor means, and determining the storing in a memory means position information corresponding to the apparent position of said sensor means at said reference time;

(b) at a time later than said reference time performing step (a) above and determining later position information corresponding to the apparent position of said sensor means at said time later than said reference time;

(c) determining a drift error correction by comparing said position information and said later position information; and

(d) utilizing said drift error correction to correct the positioning of the beam by the beam positioning means.

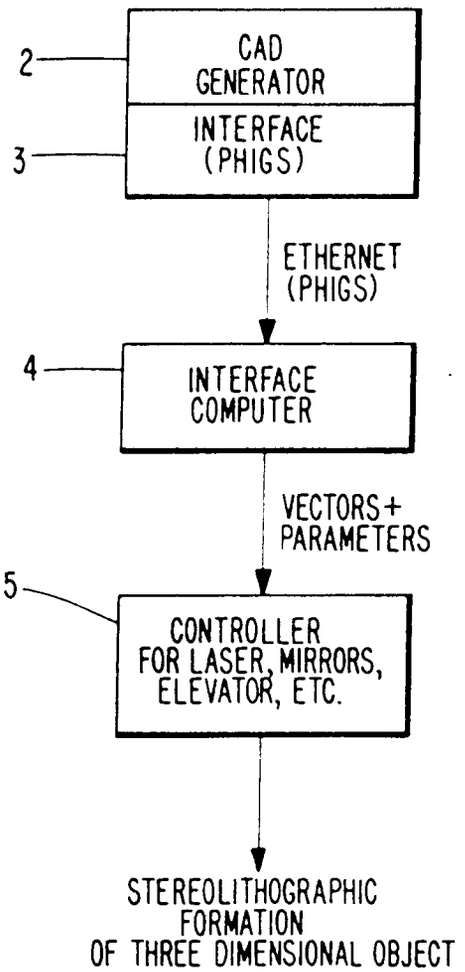

2
CAD GENERATOR
3
INTERFACE (PHIGS)
ETHERNET (PHIGS)
4
INTERFACE COMPUTER
VECTORS+ PARAMETERS
5
CONTROLLER FOR LASER, MIRRORS, ELEVATOR, ETC.
STEREOLITHOGRAPHIC FORMATION OF THREE DIMENSIONAL OBJECT

FIG. 1.

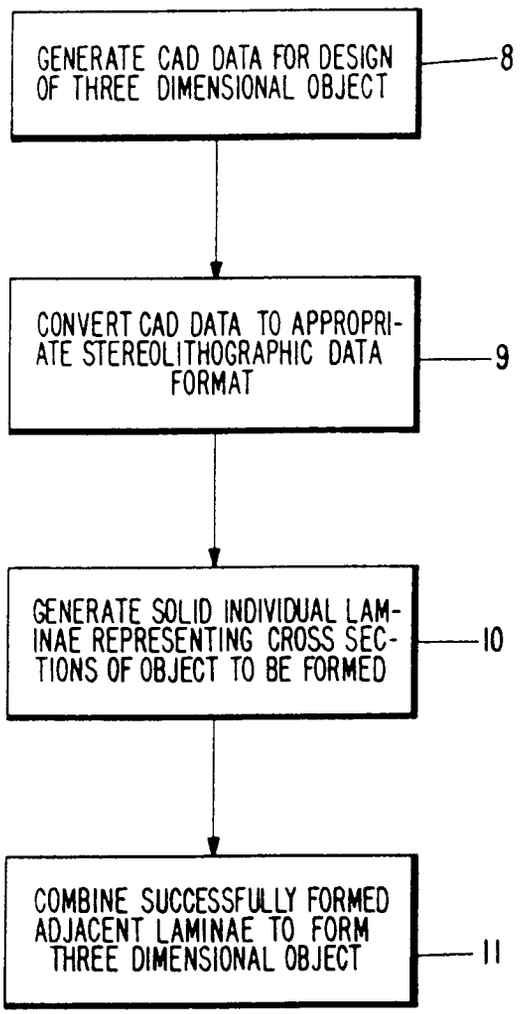

GENERATE CAD DATA FOR DESIGN OF THREE DIMENSIONAL OBJECT
8
CONVERT CAD DATA TO APPROPRIATE STEREOLITHOGRAPHIC DATA FORMAT
9
GENERATE SOLID INDIVIDUAL LAMINAE REPRESENTING CROSS SECTIONS OF OBJECT TO BE FORMED
10
COMBINE SUCCESSFULLY FORMED ADJACENT LAMINAE TO FORM THREE DIMENSIONAL OBJECT
11

FIG. 2.

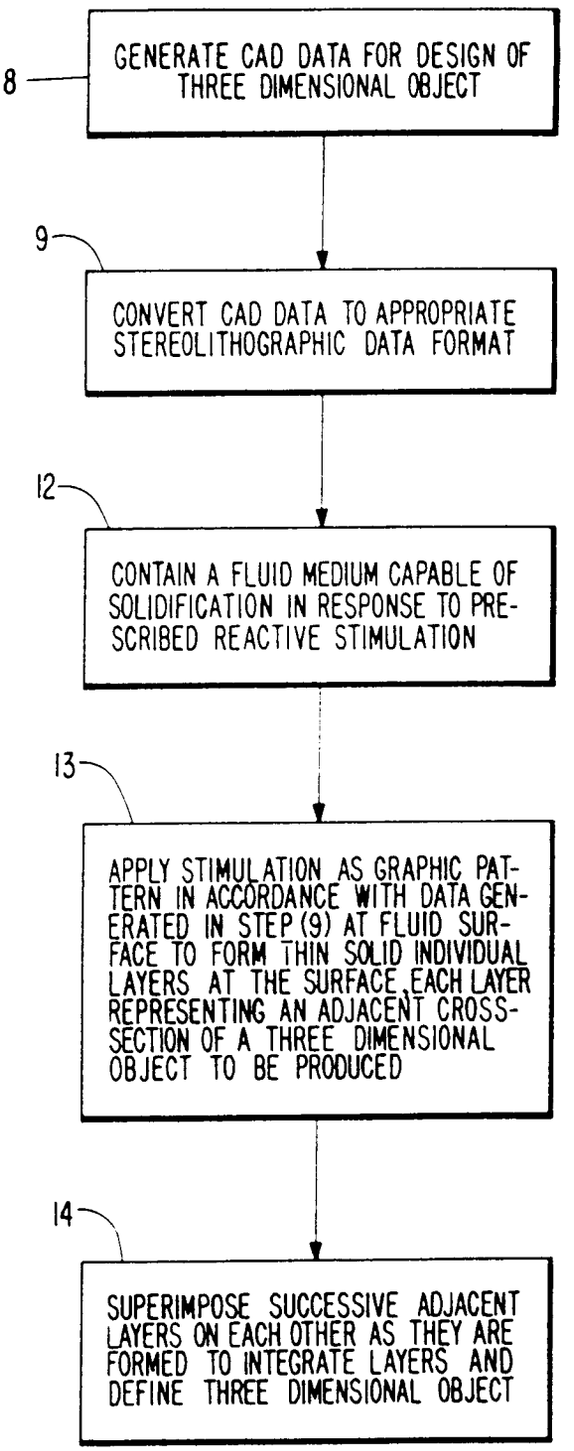
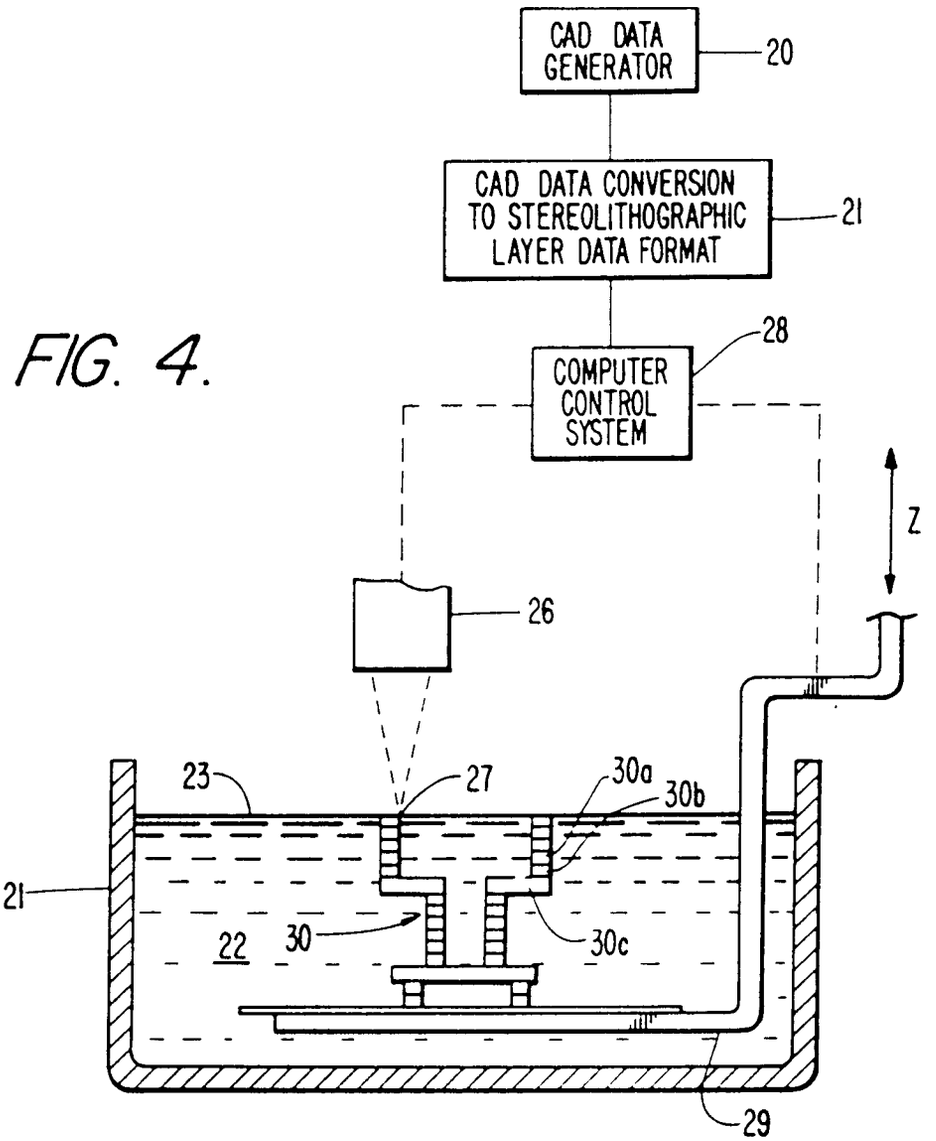

8
GENERATE CAD DATA FOR DESIGN OF THREE DIMENSIONAL OBJECT
9
CONVERT CAD DATA TO APPROPRIATE STEREOLITHOGRAPHIC DATA FORMAT
12
CONTAIN A FLUID MEDIUM CAPABLE OF SOLIDIFICATION IN RESPONSE TO PRESCRIBED REACTIVE STIMULATION
13
APPLY STIMULATION AS GRAPHIC PATTERN IN ACCORDANCE WITH DATA GENERATED IN STEP (9) AT FLUID SURFACE TO FORM THIN SOLID INDIVIDUAL LAYERS AT THE SURFACE, EACH LAYER REPRESENTING AN ADJACENT CROSS-SECTION OF A THREE DIMENSIONAL OBJECT TO BE PRODUCED
14
SUPERIMPOSE SUCCESSIVE ADJACENT LAYERS ON EACH OTHER AS THEY ARE FORMED TO INTEGRATE LAYERS AND DEFINE THREE DIMENSIONAL OBJECT

FIG. 3.

FIG. 4.


CAD DATA GENERATOR
20
CAD DATA CONVERSION TO STEREOLITHOGRAPHIC LAYER DATA FORMAT
21
28
COMPUTER CONTROL SYSTEM
Z
26
23
27
30a
30b
21
22
30
30c
29

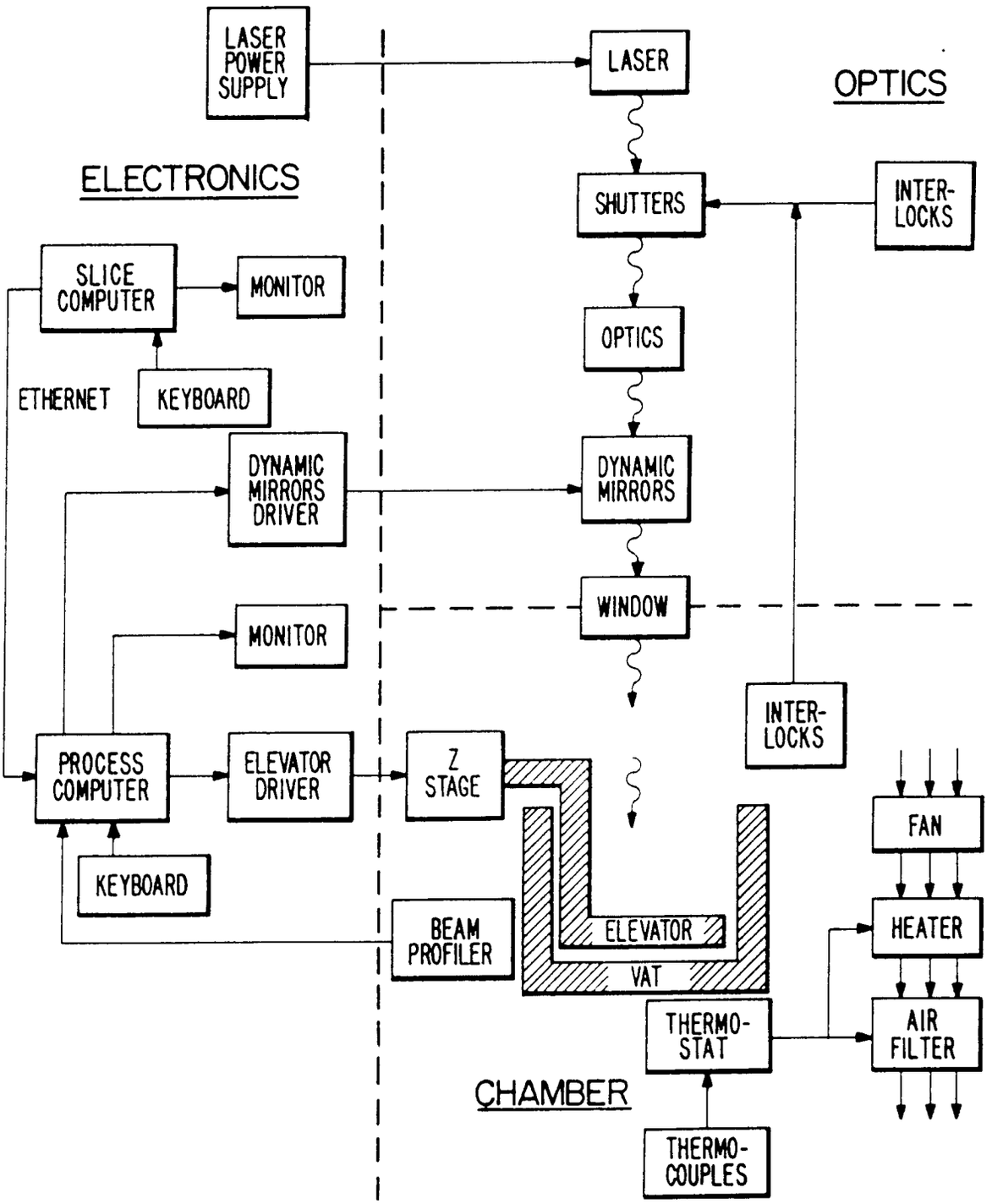

LASER POWER SUPPLY
LASER
OPTICS
ELECTRONICS
SHUTTERS
INTER-LOCKS
SLICE COMPUTER
MONITOR
OPTICS
ETHERNET
KEYBOARD
DYNAMIC MIRRORS DRIVER
DYNAMIC MIRRORS
WINDOW
MONITOR
INTER-LOCKS
PROCESS COMPUTER
ELEVATOR DRIVER
Z STAGE
FAN
KEYBOARD
BEAM PROFILER
ELEVATOR
VAT
HEATER
THERMO-STAT
AIR FILTER
CHAMBER
THERMO-COUPLES

FIG. 5.

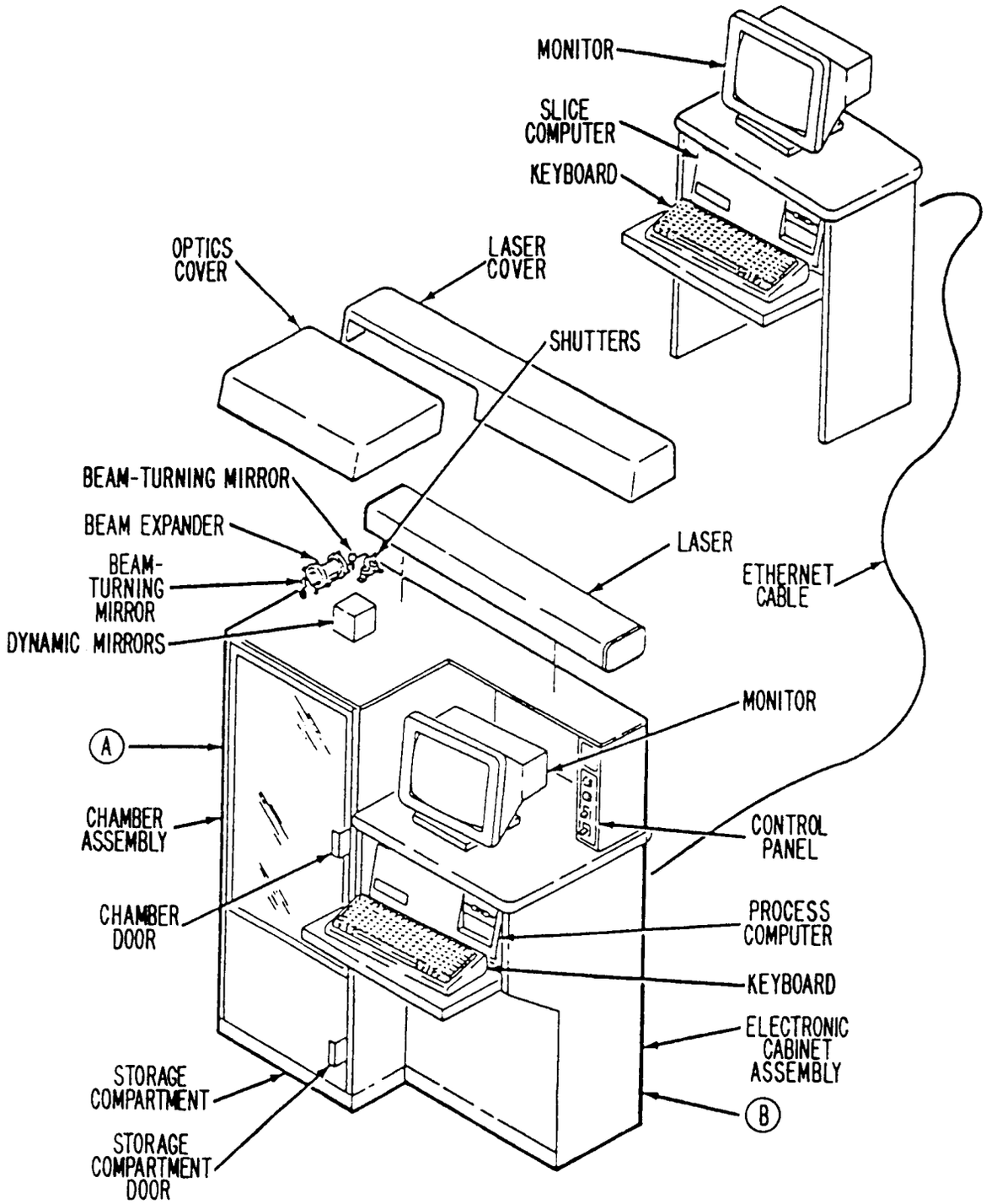

MONITOR
SLICE COMPUTER
KEYBOARD
OPTICS COVER
LASER COVER
SHUTTERS
BEAM-TURNING MIRROR
BEAM EXPANDER
BEAM-TURNING MIRROR
DYNAMIC MIRRORS
LASER
ETHERNET CABLE
MONITOR
A
CHAMBER ASSEMBLY
CONTROL PANEL
CHAMBER DOOR
PROCESS COMPUTER
KEYBOARD
ELECTRONIC CABINET ASSEMBLY
STORAGE COMPARTMENT
STORAGE COMPARTMENT DOOR
B

FIG. 6.

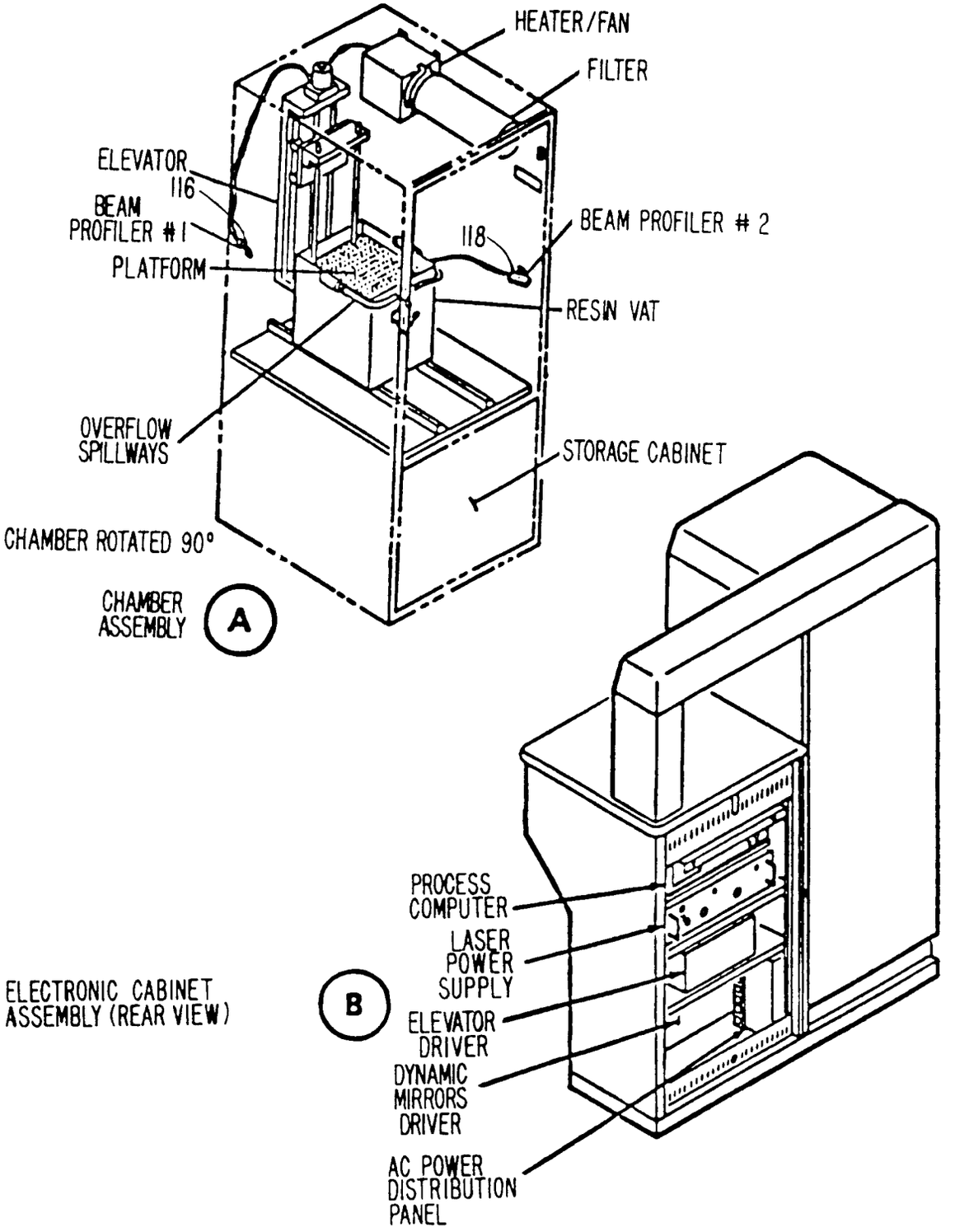

HEATER/FAN
FILTER
ELEVATOR
116
BEAM
PROFILER #1
PLATFORM
118
BEAM PROFILER # 2
RESIN VAT
OVERFLOW
SPILLWAYS
STORAGE CABINET
CHAMBER ROTATED 90°
CHAMBER
ASSEMBLY
A
PROCESS
COMPUTER
LASER
POWER
SUPPLY
ELECTRONIC CABINET
ASSEMBLY (REAR VIEW)
B
ELEVATOR
DRIVER
DYNAMIC
MIRRORS
DRIVER
AC POWER
DISTRIBUTION
PANEL

FIG. 7.

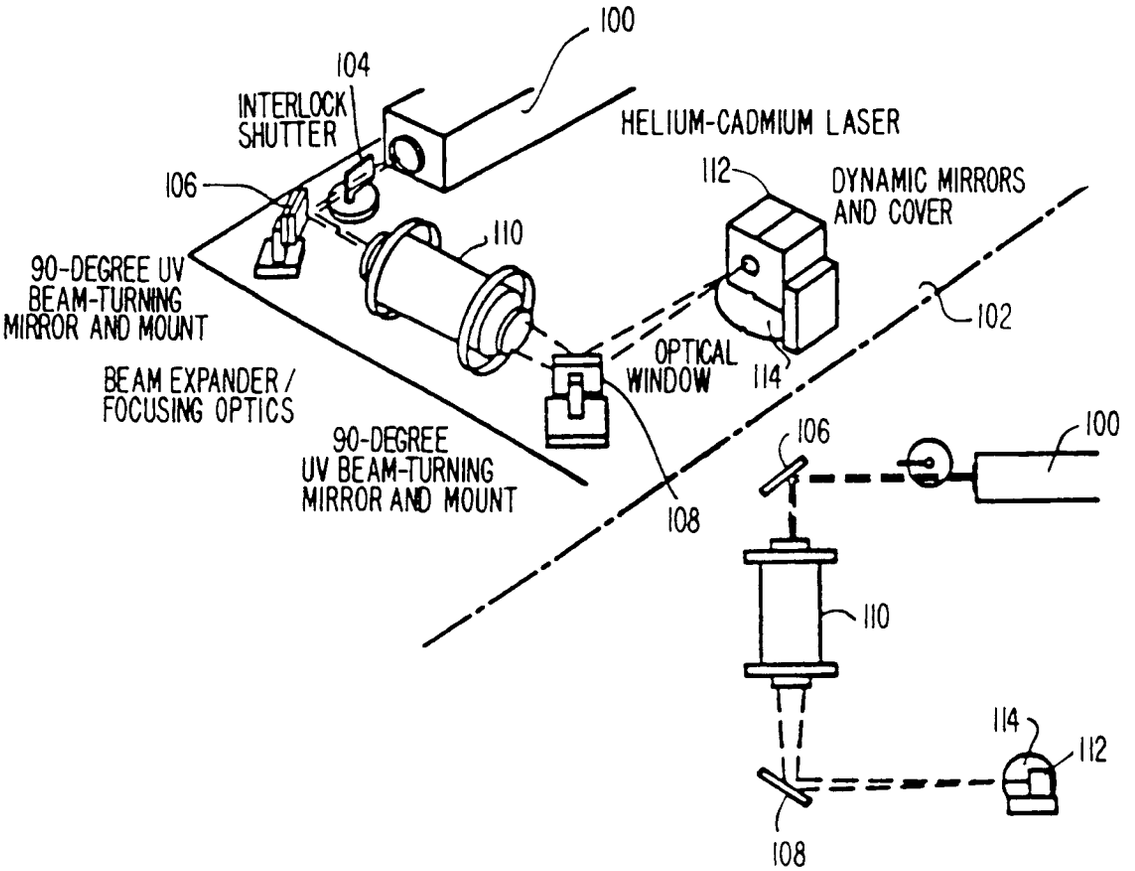

100
104
INTERLOCK SHUTTER
HELIUM-CADMIUM LASER
106
112
DYNAMIC MIRRORS AND COVER
110
90-DEGREE UV BEAM-TURNING MIRROR AND MOUNT
102
BEAM EXPANDER / FOCUSING OPTICS
OPTICAL WINDOW
114
90-DEGREE UV BEAM-TURNING MIRROR AND MOUNT
108
106
100
110
114
112
108

FIG. 8.

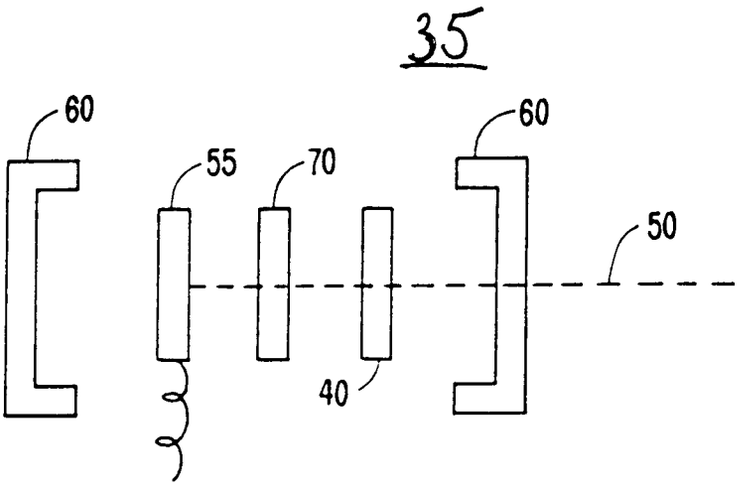


FIG. 9a.

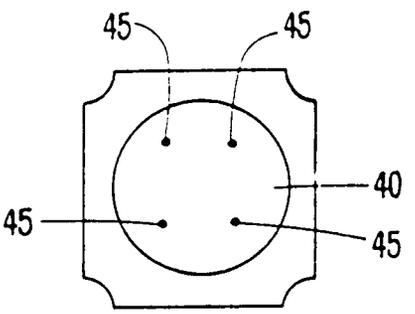


FIG. 9b.

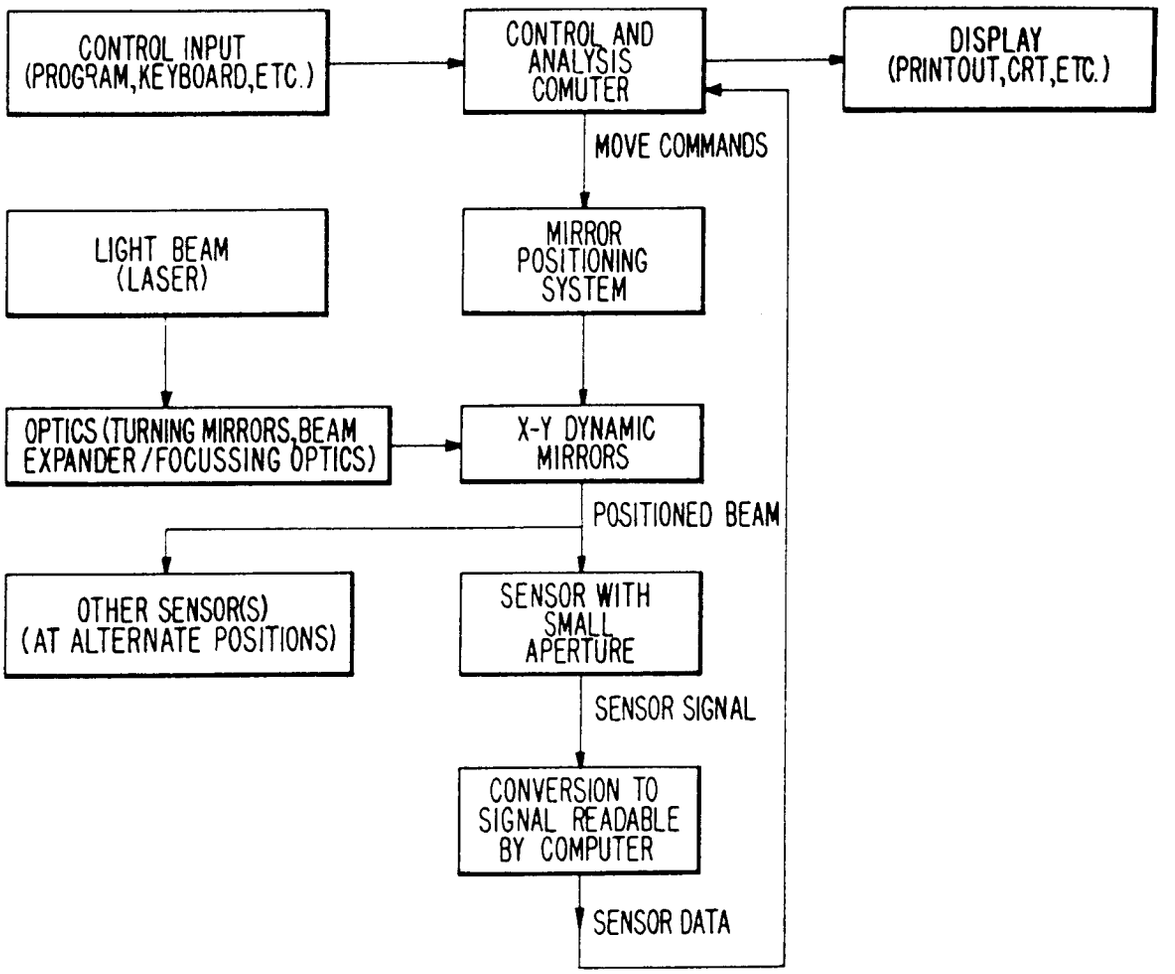

CONTROL INPUT (PROGRAM,KEYBOARD,ETC.)
CONTROL AND ANALYSIS COMUTER
DISPLAY (PRINTOUT,CRT,ETC.)
MOVE COMMANDS
LIGHT BEAM (LASER)
MIRROR POSITIONING SYSTEM
OPTICS (TURNING MIRRORS,BEAM EXPANDER/FOCUSSING OPTICS)
X-Y DYNAMIC MIRRORS
POSITIONED BEAM
OTHER SENSOR(S) (AT ALTERNATE POSITIONS)
SENSOR WITH SMALL APERTURE
SENSOR SIGNAL
CONVERSION TO SIGNAL READABLE BY COMPUTER
SENSOR DATA

FIG. 10.

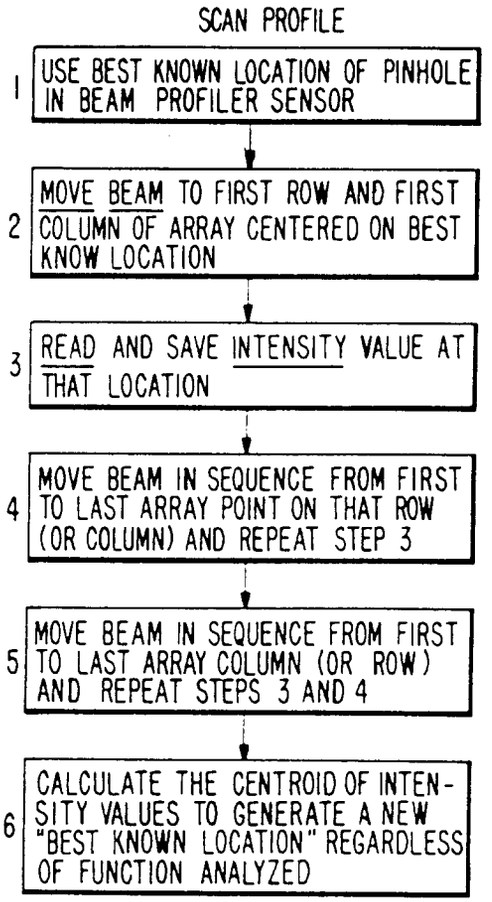


FIG. 11a.

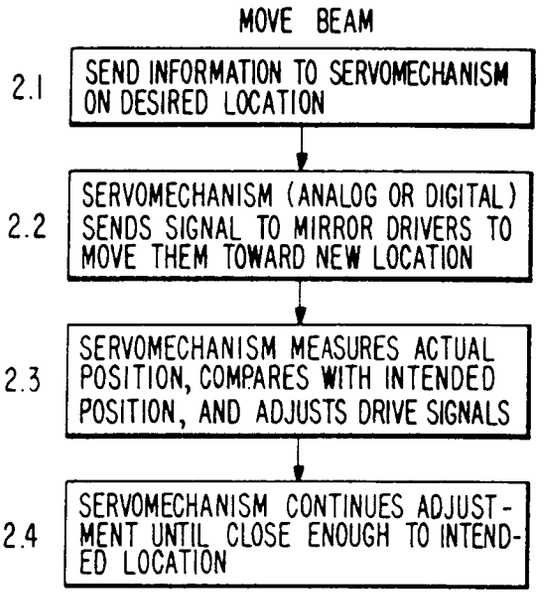


FIG. 11b.

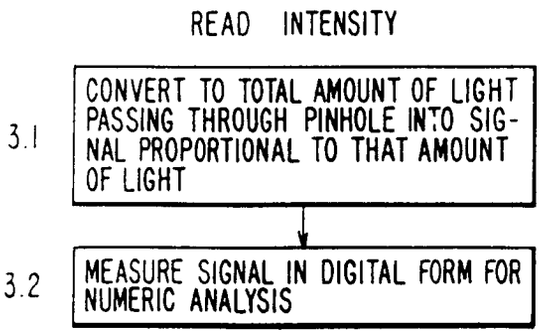


FIG. 11c.

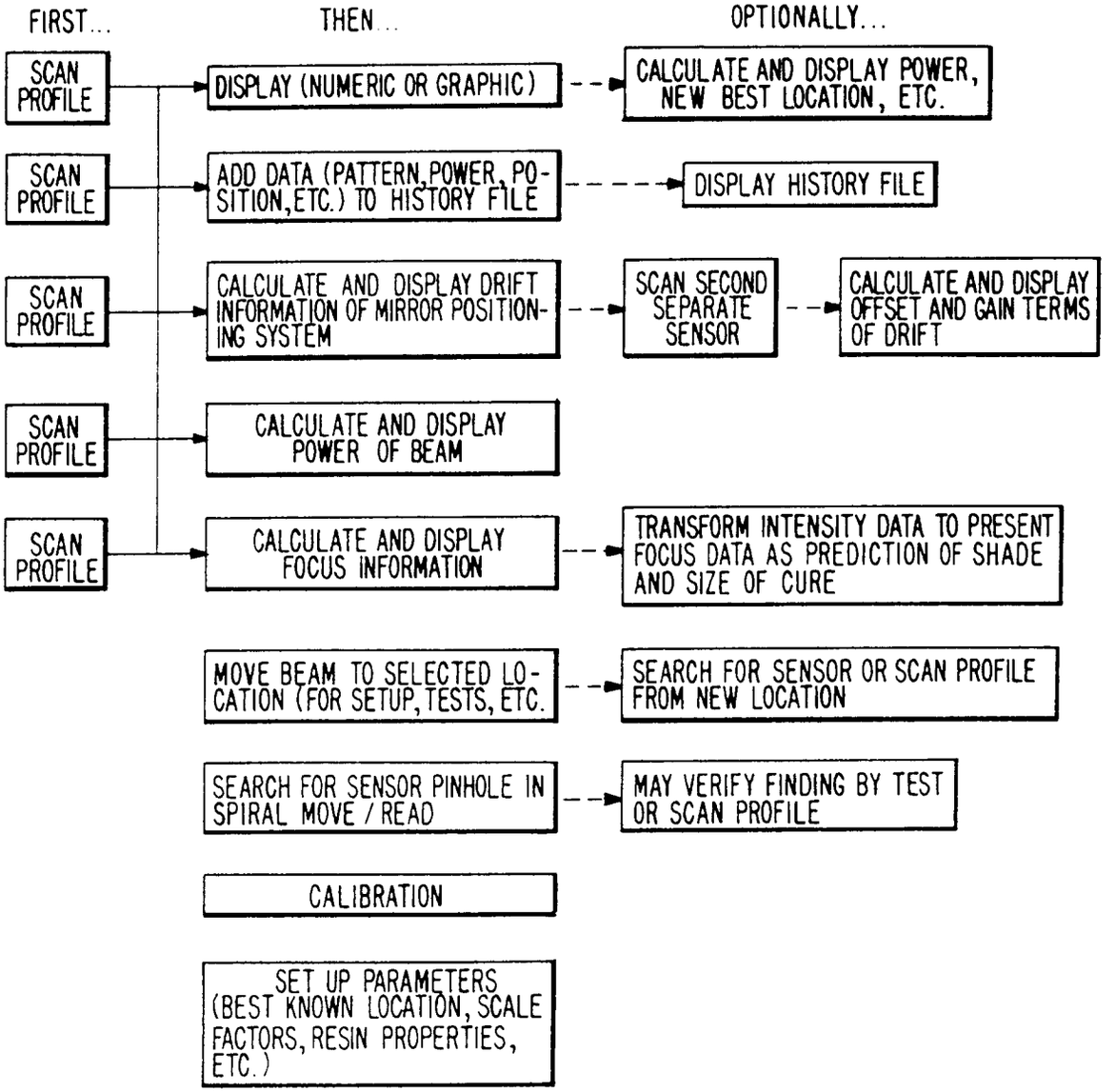

FIRST...
THEN...
OPTIONALLY...
SCAN PROFILE
DISPLAY (NUMERIC OR GRAPHIC)
CALCULATE AND DISPLAY POWER, NEW BEST LOCATION, ETC.
SCAN PROFILE
ADD DATA (PATTERN, POWER, POSITION, ETC.) TO HISTORY FILE
DISPLAY HISTORY FILE
SCAN PROFILE
CALCULATE AND DISPLAY DRIFT INFORMATION OF MIRROR POSITIONING SYSTEM
SCAN SECOND SEPARATE SENSOR
CALCULATE AND DISPLAY OFFSET AND GAIN TERMS OF DRIFT
SCAN PROFILE
CALCULATE AND DISPLAY POWER OF BEAM
SCAN PROFILE
CALCULATE AND DISPLAY FOCUS INFORMATION
TRANSFORM INTENSITY DATA TO PRESENT FOCUS DATA AS PREDICTION OF SHADE AND SIZE OF CURE
MOVE BEAM TO SELECTED LOCATION (FOR SETUP, TESTS, ETC.
SEARCH FOR SENSOR OR SCAN PROFILE FROM NEW LOCATION
SEARCH FOR SENSOR PINHOLE IN SPIRAL MOVE / READ
MAY VERIFY FINDING BY TEST OR SCAN PROFILE
CALIBRATION
SET UP PARAMETERS (BEST KNOWN LOCATION, SCALE FACTORS, RESIN PROPERTIES, ETC.)

FIG. 12.

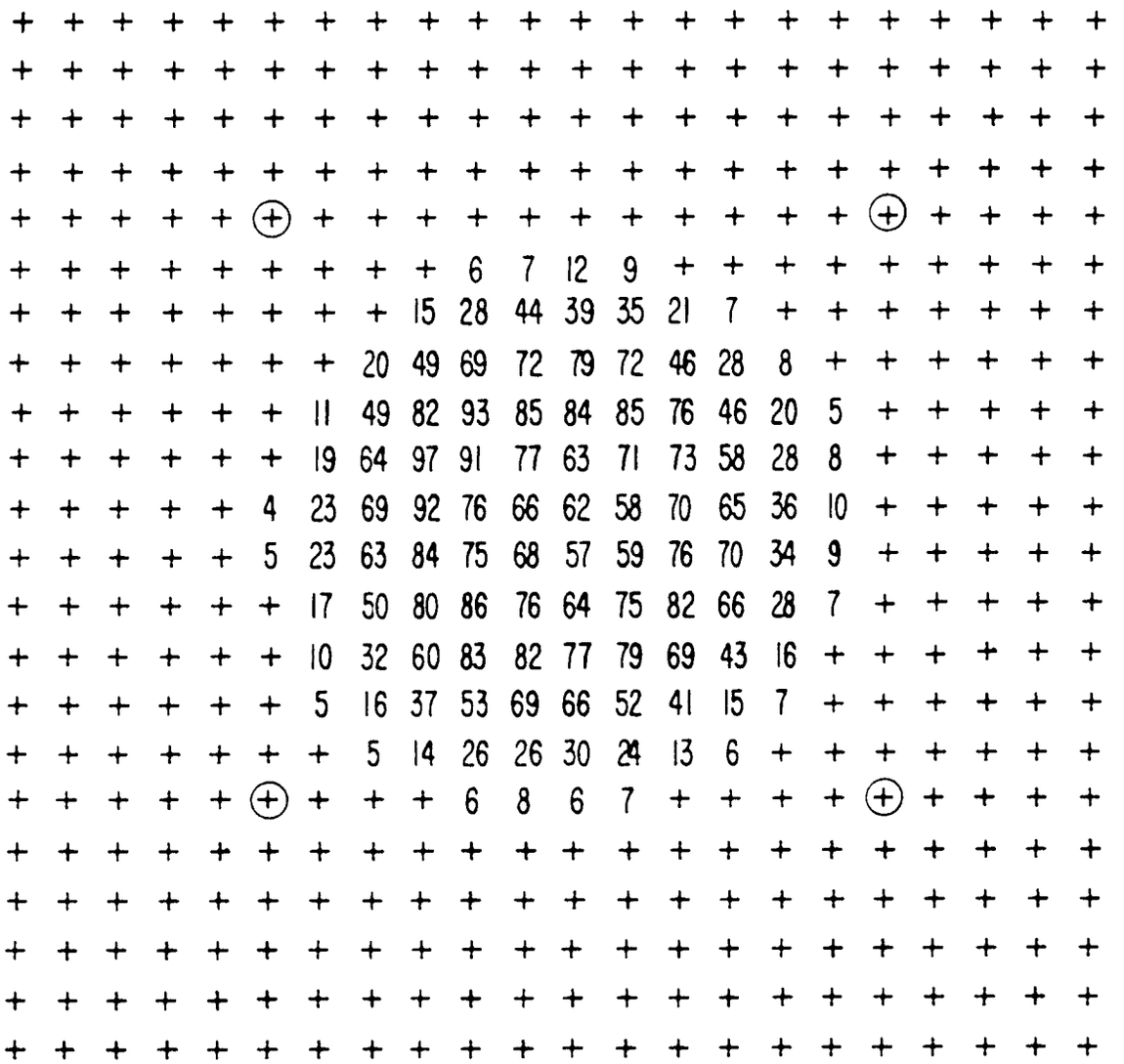

6 7 12 9
15 28 44 39 35 21 7
20 49 69 72 79 72 46 28 8
11 49 82 93 85 84 85 76 46 20 5
19 64 97 91 77 63 71 73 58 28 8
4 23 69 92 76 66 62 58 70 65 36 10
5 23 63 84 75 68 57 59 76 70 34 9
17 50 80 86 76 64 75 82 66 28 7
10 32 60 83 82 77 79 69 43 16
5 16 37 53 69 66 52 41 15 7
5 14 26 26 30 24 13 6
6 8 6 7

FIG. 13.

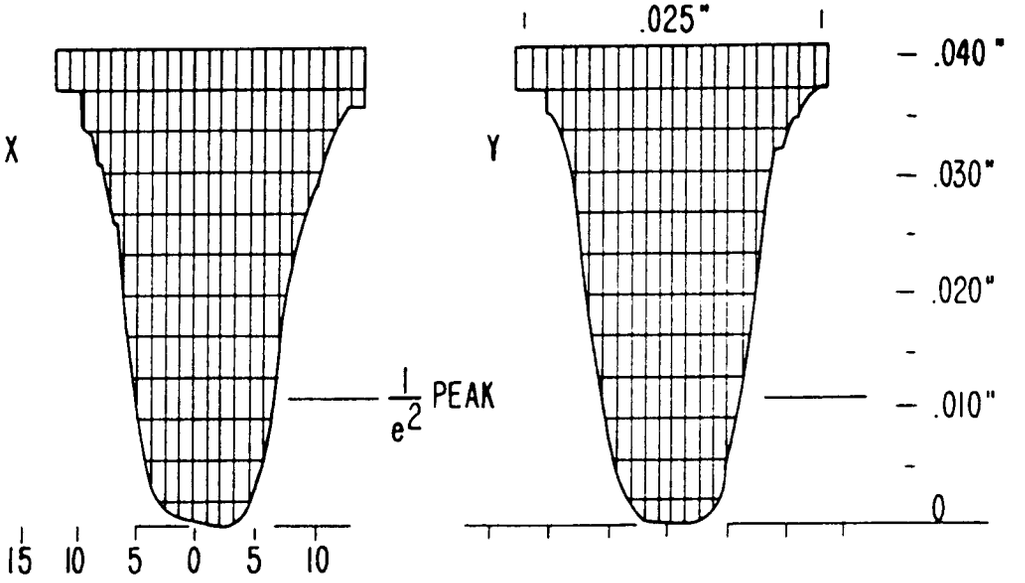
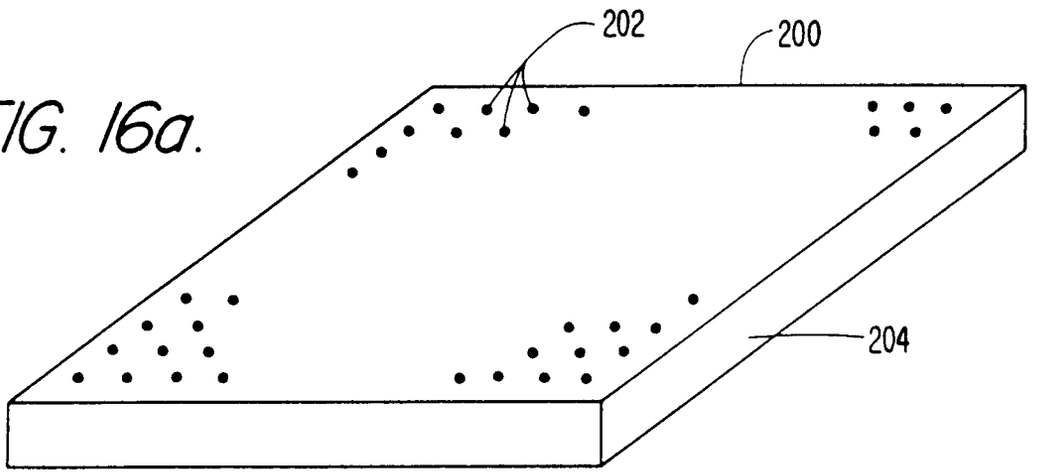


FIG. 14.



FIG. 15.

FIG. 16a.



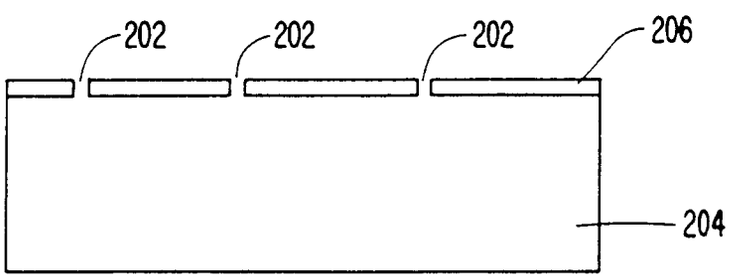


FIG. 16b.